
# OCON

## A context-awareness framework

Jacob B. Cholewa
&
Mathias K. Pedersen

May 21, 2014

# Contents

# 1 Introduction

Ubiquitous computing is an area of computer science, coined by Mark Weiser in the early nineties, it is the vision of weaving technology into the fabric of everyday life. Context-awareness is a cornerstone of ubiquitous computing, which brings situational information from the physical environment, the real world, into applications which is essential for realizing ubiquitous computers. This project presents a framework for aiding developers in making context-aware applications.

## 1.1 Project description

A Context-aware system is able to adapt its behavior to its surroundings. In order to act upon its environment, the system will need sensor-input. Various sources of input can be used to determine the actions of the system.

In this project we want to mainly focus on implementing a context-awareness framework, making it easy for systems to actuate on sensor events. As a proof of concept we will explore replacing the physical Scrum board, which is often a whiteboard and post-its, with an IT-solution which is not confined to a personal computer, but has the same presence. The digital Scrum board will be context-aware to the extent that it can recognize different Scrum activities, like sprint meeting or one-on-one-meeting, and automatically change its graphical interface.

## 1.2 Scope of this project

As per our Project description there are two artifact to be made; The context awareness framework and a proof of concept implementation in form of a Context-aware Scrum board.

**The Framework.** Our project description dictates that the main piece of work will be our framework, and so implementation of this framework will have the broadest scope.

**Goal 1** *Our aim is to implement a framework that is adaptable and easy to use.*

**The Scrum board.** For us to prove usability of the framework developed we will implement it for a context-aware Scrum board. The function of this artifact is solely to test the usefulness of the context-awareness framework.

**Goal 2** *In trying to prove Goal 1, implement a stub Scrum board utilizing the framework*

It is not in our scope to implement a functional or useful Scrum board.

# 2 Background Research

## 2.1 Context awareness

To be able to make a context-aware framework, we first had to investigate what context and context-awareness is in a computer science perspective.

Context-awareness is a term associated with Ubiquitous computing. Ubiquitous computing, ubicomp, was coined in the early nineties by Mark Weiser whose vision was to make technology that could seamlessly assist in everyday tasks. Weiser's research-unit at Xerox PARC developed some of the first mobile devices, and the development of ubi computing clearly reflects in todays technology boom of smart phones and tablets.

> *"We have found two issues of crucial importance: location and scale. Little is more basic to human perception that physical juxtaposition, and so ubiquitous computers must know where they are. (Today's computers, in contrast, have no idea of their location and surroundings.) If a computer knows merely what room it is in, it can adapt its behavior in significant ways without requiring even a hint of artificial intelligence."* Mark Wiser (1991) [1]

As Weiser and his research team realised: For a system to weave itself into the everyday life to seamlessly interact with humans, the system must acquire knowledge to the current situation or *context*. To make succeeding Ubiquitous computers, information of the surrounding environment is essential. Context-awareness provides environmental information, *context-information*, to applications so that they can adapt a behaviour suitable to the current *context*. Many of the publications on the subject describe context differently, but the one description fitting best our understanding was coined by Dey and Abowd whom described context as:

> *"Any information that can be used to characterize the situation of an entity. An entity is a person, place or object that is considered relevant to the interaction between a user and an application, including the user and application themselves."* [2]

The holy grail of context-awareness is to understand and perform human intent, but it is important to keep in mind how difficult human intent is to understand. A person himself might even experience wonder at his own intentions. When humans interact, they can interpret the ongoing situation from the implicit understanding of body language, tone of voice, the surrounding environment as well as previous relations and information about past and future events. These *context informations* are

essential factors in effective communication.

Human-to-human interaction is, although the tremendous amount of context-information involved, never perfect and factors like culture differences have thought us the importance of knowledge in understanding intentions. The same goes for human-machine interactions where it seems a large amount of context-information, along with the ability to interpret it, is necessary for a machine to understand complex human intentions.

This said, not only increasing the available amount of context-information, but also improving the quality and correctness of the information is important. As the amount of context increases, the context-aware application becomes able to take actions without explicit user input. When doing so, action can be taken on wrong or typically incomplete snaps of context.

> *"Intelligibility and control are important user concerns in context-aware applications. They allow a user to understand why an application is behaving a certain way, and to change its behaviour."* [3]

The above cited article stresses that insight to why an implicit action is carried out is important, and that it is important to offer the user the possibility to overrule the action. This brings us back to the previous discussed topic of understanding human intent. Computers will never fully understand human intent and therefore it is important to understand when implicit actions might be erroneous and the user therefore should be prompted before the action or have the ability to change the state back again.

## 2.2 Context aware frameworks

To support application developers, a number of context-frameworks have been developed, for example JCAF and Context toolkit. When looking at previously developed frameworks mainly two approaches were used: Blackboard and Widget-based architecture [5]

The blackboard approach is a centralized solution. Sensors and applications are connected to the blackboard and whenever a new sensor state is available a post-it, an entry to the database, is put on the blackboard. The application can at will look through the blackboard and search for context it might find relevant. The blackboard abstracts away the sensor implementation allowing the client to focus on only the context information.

The widget approach is an object-oriented distributed architecture. Sensors encapsulated by widgets are available for application subscription. The approach is event based and applications are notified whenever a change to the sensor is occurring. This approach is object-oriented as the context is modelled with objects sent from sensor to application. This time and space coupled solution stands in contrast to the blackboard approach which has a database and is therefore time- and space uncoupled.

The approaches differ a lot in the way of modelling context, but the main difference is in the way they deliver context from sensor to application. The stand in great contrast when looking at space and time coupling.

Where the blackboard at any time offers applications to go though its context database, it does

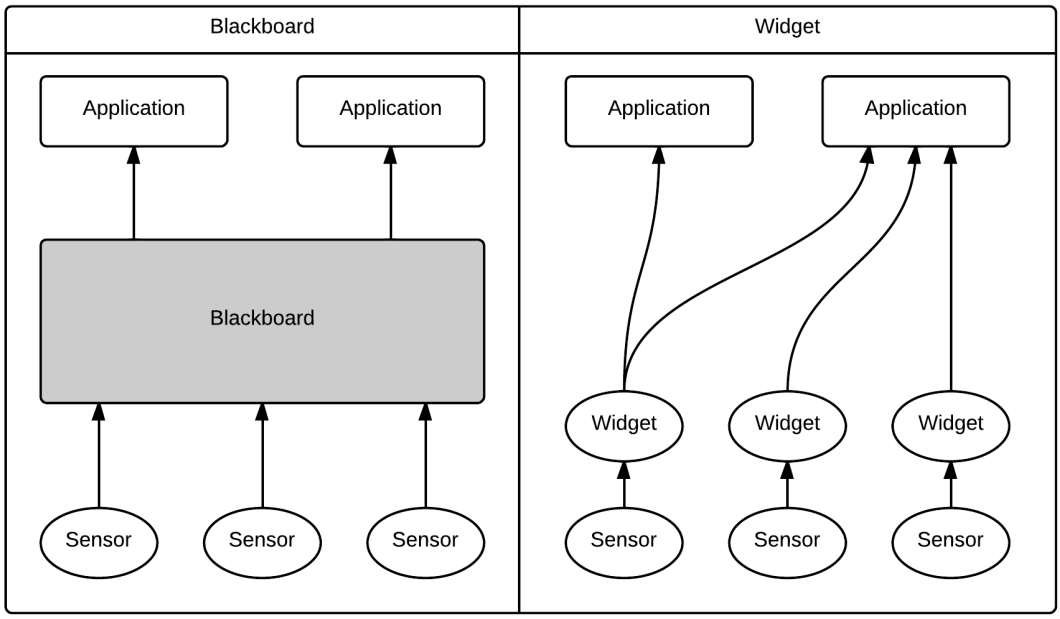


Figure 2.1: Blackboard and widget-based approach

not offer notifying the application, as the blackboard is space uncoupled, and does not know about the application. The widget solution offers live updates only when they happen, and only to the applications subscribing for the update. Both architectures are very useful in different applications.

For example a hospital system where you want to use the context framework to track patients. The blackboard solution seems to meet requirements best as you can, when needed, look up a patients whereabouts in the hospital. This example requires lookup and that all information is gathered in one place, supported by the blackboards time uncoupling and centralization. The traditional blackboard solution does on the other hand not support to warn for example the closest employee if a patient is trying to escape the premises which would be very useful.

For a home automation system using sensors, sensor input is only interesting the moment it happens. When a person enters the room the light should go on, only in the room where the person entered, and only at the given time. This example needs the time and space coupling supported by the widget and would not be doable with the tradition blackboard architecture.

# 3 Design

This chapter describes the different design choices considered while developing the framework.

## 3.1 Architecture

When looking at the blackboard and widget based methods, we have decided that we want to include a little of both.

We want to do a centralized system where clients can register a situation predicate and our implementation, the Ocon framework, will be event-based. Sensors encapsulated by widgets will provide sensor input. The central will evaluate the predicate and whenever the state changes, an event will be fired to notify the client. The client will be the application's entry point to Ocon.

This combines the blackboards centralization and abstraction with the widgets object-oriented modelling and time and space coupling. This will result in a solution where it's transparent for the developer which sensor is actually given the input, like the blackboard, but also having the space and time coupling from the widget based method.

With this Ocon will be an If-this-then-that solution where the client can use sensor input for control without the client developers having to put much though into using and managing sensors. This will be realized by three different components. A client, a central and a widget (See figure 3.1).

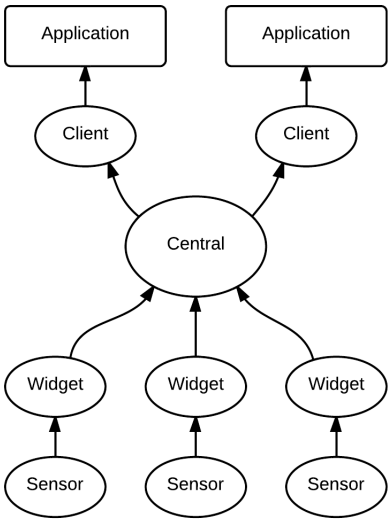


Figure 3.1: Architecture of Ocon

## 3.2 Encapsulation of Context-information

To best support the overall architecture, doing an object-oriented encapsulation has been chosen. Object-oriented encapsulation of context can be done in a variety of ways. We have been looking at two different methods. Using a composite pattern and modelling with properties. In this chapter we will examine the two methods.

Using a composite pattern enforces relations between entities, entities being locations, persons, things or other real world objects. Lets say we want to model a location with some rooms, with some persons, with different items. Using the composite modelling we can have a location object containing room objects containing person objects containing item objects. We will then have a relationship between the objects compositing the context being that a group of students is are room 3A04 at ITU all having phones in their pockets and all sitting down. This is a fairly complex, but a very extensible and flexible way to model situation (See fig 3.2). The downside to this method is that you can easily have an overflow of objects, and that it can be very complex and computation heavy to check against a predicate.

The other method we are considering is more simple. Having a set of entities that we wish to track, entities again being locations, persons or things. All context information relating to a person will be properties to the object. So to use the previous example, a person entity would then have a location property, being ITU room 3A04, a phone property begin true and a sitting property being true. This model is more easy to make, but it limits the developer to know very precisely what information is needed and should be tracked. The previous model has the advantage of being very flexible allowing more complex relations and situations.

Both methods have pros and cons being how dynamic they are, how easily they can be implemented, and what performance they will have and that will be the parameters we'll look at when implementing Ocon

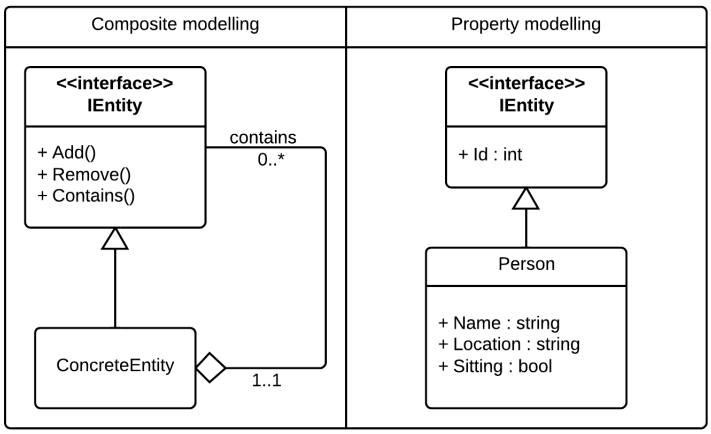


Figure 3.2: Diagram illustrating composite vs property modelling

## 3.3 Central

The central is the core component and will facilitate clients adding predicates for the central to track. When an entity change relevant to the predicate is received, the predicate will be checked and if its state changes, the owning client will be notified.

The definition of predicate is taken from the .NET platform. A predicate is a delegate returning a boolean. In this way .NET developers will be familiar with the meaning and definition of a predicate.

An important feature will be the central's ability to be discovered by client and widgets. The central will be able to broadcast itself so widgets and clients can discover and connect to the central. This feature will enforce a more dynamic usage and behaviour of Ocon. Dynamic behaviour being to have a running central where clients and widgets are able to connect and disconnect without having to restart or make any actions to the central.

## 3.4 Widget

The widgets will translate sensor input to entities along with facilitating communication to the central. This can be seen in figure 3.3

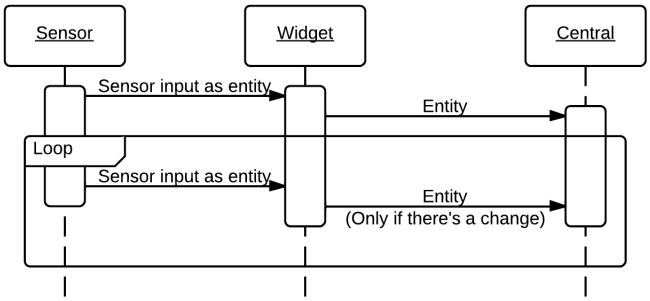


Figure 3.3: Information flow from Widget to Central

This approach has been chosen to make the system easily distributed as a small, maybe embedded, component can run a widget translating the sensors input to entities before sending them to the central. This approach will also reduce network load as the widget will only send updates to the central when changes occur. This stands in contrast to sending all sensor input to the central and making it process the data itself which could be useful in some cases. Sending all data would also put more computational pressure on the central making it hard to scale to larger systems. Last but not least the widget will have an architecture making it convenient and easy to use for developers. This will be done to pursue *goal 1*

## 3.5 Client

The client will be the entry point into Ocon facilitating addition of situations to the central, and receive updates whenever an update to those situations occur. The Objectives of the client is to automatically discover and use the central and as with the widget, have an architecture making it easy and convenient for developers to use. Figure 3.4 shows the basic idea for the information flow from central to client and application.

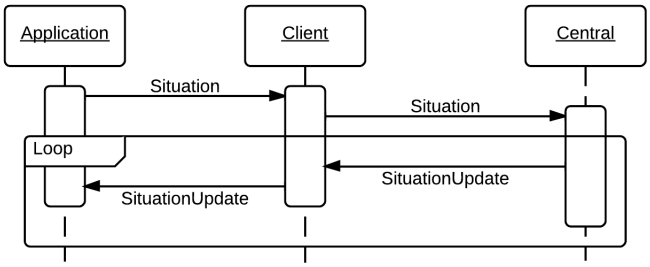


Figure 3.4: Information flow between client and central

## 3.6 Communication

Clients and widgets will be distributed from the central. The reason for doing so is that our vision of having a centralized system distributing situation from sensors to clients is not very useful if not distributed.

For Ocon two different communication protocols need to be implemented.

- A protocol for establishing link between peers
- A protocol for send text/json messages for subscriptions, subscription events and, sensor events

We do not wish to bind our users to any concrete communication protocols. Therefore the communication will be interfaces so a concrete implementation can be dependency injected into Ocon.

# 4 Implementation

## 4.1 Overview

This chapter will describe important implementation details of Ocon. The system is realized by three main components. A OconCentral, a OconWidget and a OconClient. In addition a communication library have been created to facilitate distributed communication between the three components.

## 4.2 Encapsulation of Context

Entities are generalized by the interface IEntity, specifying general information for all entities. AbstractEntity extends on IEntity and overrides ToString() with a meaningful implementation. Both are public and it is up the an implementer which to extend.

On figure 4.1 is an example of Person extending AbstractEntity with a present property motivated by the need for this context information.

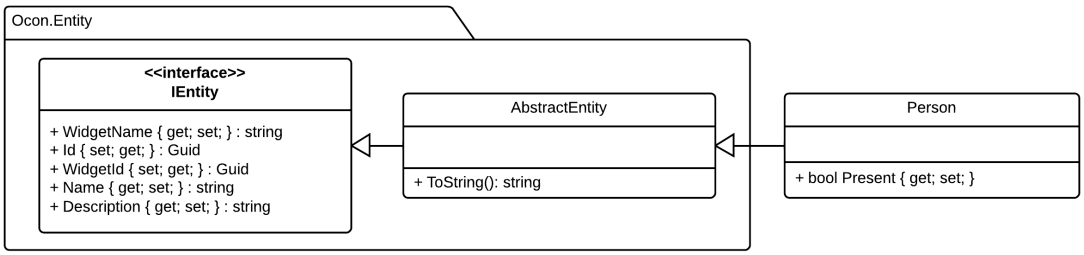


Figure 4.1: Person as a custom IEntity implementation

## 4.3 Communication

To support the design decision that the communication module should be injectable a communication interface, IOconCom, have been implemented. (The interface can be seen in figure 4.5 on page 16)

With this design developers can make their own implementation for communication. Ocon contains a default IOconCom implementation build using the TCP/IP layer. For serialization we have chosen to use JSON. These decisions have been made so that developers are not bound to the .NET platform and can make clients and widgets in other languages like Java, or C directly on a micro controllers like Arduino.

For peer discovery we have chosen to use IP multicast. The central broadcasts itself for peers to discover. The peers, OconClients and OconWidgets, are listening on the multicast endpoint and will invoke an event when a central is discovered.

To facilitate discovery OconTcpCom implements two methods: Broadcasting and DiscoveryService. Both methods are running on threads. This is done so that that peers are not blocked when broadcasting or discovering. When a peer is discovered an DiscoveryService event will be fired to notify subscribers.

For sending Entites, situation updates and situation subscription following methods have been implemented: Listen, SendEntity, SendSituationState, and SubscribeSituaion. When listen is called a new thread is started and whenever a message is received either IncommingSituationSubscribtion-Event, IncommingEntityEvent, or IncommingSituationChangedEvent is fired to notify subscribers. The different send methods are also executed on a thread. This has been done to avoid blocking and to get faster execution time as multiple messages can be sent in parallel

The send methods take a peer which is a helper class. Just like entities, peers are assigned a GUID. The GUID is a unique identifier for the peer and the communication class stores the IPEndPoints associated with the GUIDs. When a message from a peer is received or discovered the IpEndPoint is saved in a HashSet. By doing this messages can be sent only by using the peer and not the IPEndPoint which adds an abstraction layer to the module.

## 4.4 OconCentral

OconCentral is the core component of Ocon. The central contains a OconContextFilter and a concrete implementation of IOconCom. The context filter is aggregated of entities and situations (see figure 4.5 on page 16). The implementation of a situation contains a predicate that given a set of entities can evaluate to true or false. The context filter can therefore by its set of entities evaluate the state of its situations.

OconCentral subscribes to the IncommingSituationSubscribtion event and when invoked the central will add the requesting peer to the given situation. It also subscribes the IncommingEntity event on IOconCom. When invoked the entity is added to the context filter and the context filter will check its predicates. If a situation changes a SituationChanged event will be fired. The central

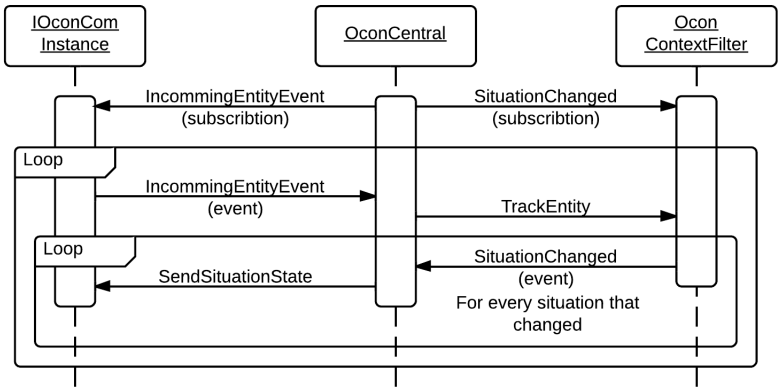


Figure 4.2: Central sequence diagram

subscribing this event will when invoked call SandSituationChanged on IOconCom to notify all subscribers on the situation of its new state. This is shown in figure 4.2

## 4.5 OconWidget

The OconWidget's purpose is to track entities and keep them updated in the central. The widget developer should translate sensor input to IEntity implementations and then pass them to the OconWidget which will facilitate sending them to the OconCentral (See figure 4.3)

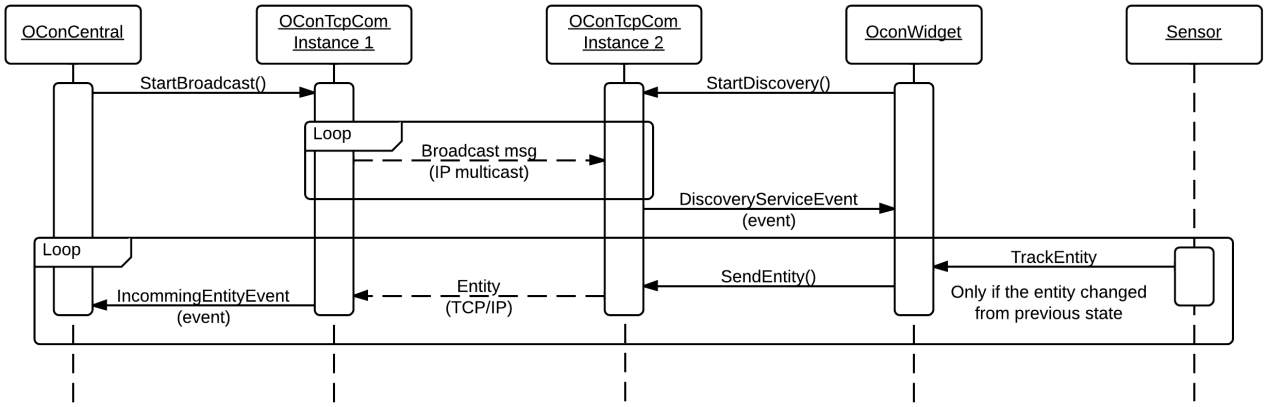


Figure 4.3: Widget to Central sequence diagram

The OconWidget is very light. When constructed the widget will listen for OconCentrals by starting discovery and subscribing to DiscoveryServiceEvents on the IOconCom. When a central is found an event will be fired and the central will be added as subscriber. OconWidget has only one method. Notify(IEntity entity). When Notify is called from the sensor a check weather or not the entity is already tracked is performed. If it is not tracked it will be allocated a new GUID before it is sent to the central through the communication module. The GUID is a unique ID used for distributed systems and allows Ocon to distinguish between adding the entity as a new entity or updating an entity already know to the framework.

## 4.6 OconClient

The OconClient's purpose is to send predicates to the central for tracking. When a situation update is send from the central the client must be able to notify the parent application about the situation update. (See figure 4.4)

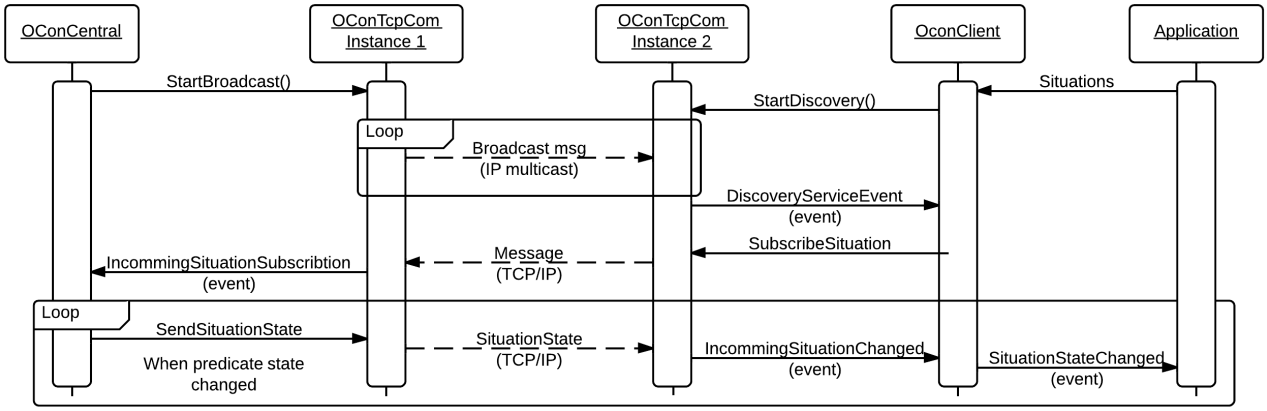


Figure 4.4: Client to Central sequence diagram

The OconClient is, as the OconWidget, very light. It contains only a constructor and a SituationStateChanged Event. The constructor takes a list of predicate identifiers as parameter, subscribes to DiscoveryServiceEvents on IOconCOm, and starts the discovery service. When a DiscoveryServiceEvent is invoked the OconClient sends its predicate identifiers to the OconCentral.

Per design the client was supposed to send a predicate and not a predicate identifier. To support clients being able to send predicates to the central, predicates must be serialized, but this is unfortunately not supported in .NET. Frameworks for handling the serialization have been researched, but none of the investigated frameworks were able serialize the predicates in Ocon as custom types are used. Therefore serialization of predicates was forced out of scope. Instead of sending the predicates, they are coded in the central. The OconClient can then subscribe to them by sending a predicate identifier.

The OconCentral will upon receiving the predicate identifiers send back the state of the subscribed predicates. When OconClient was constructed it also subscribed IncommingSituationChanged event in the communication module. When such an event is fired OconClient will fire its SituationStateChanged event notifying subscribing applications of the predicate change.

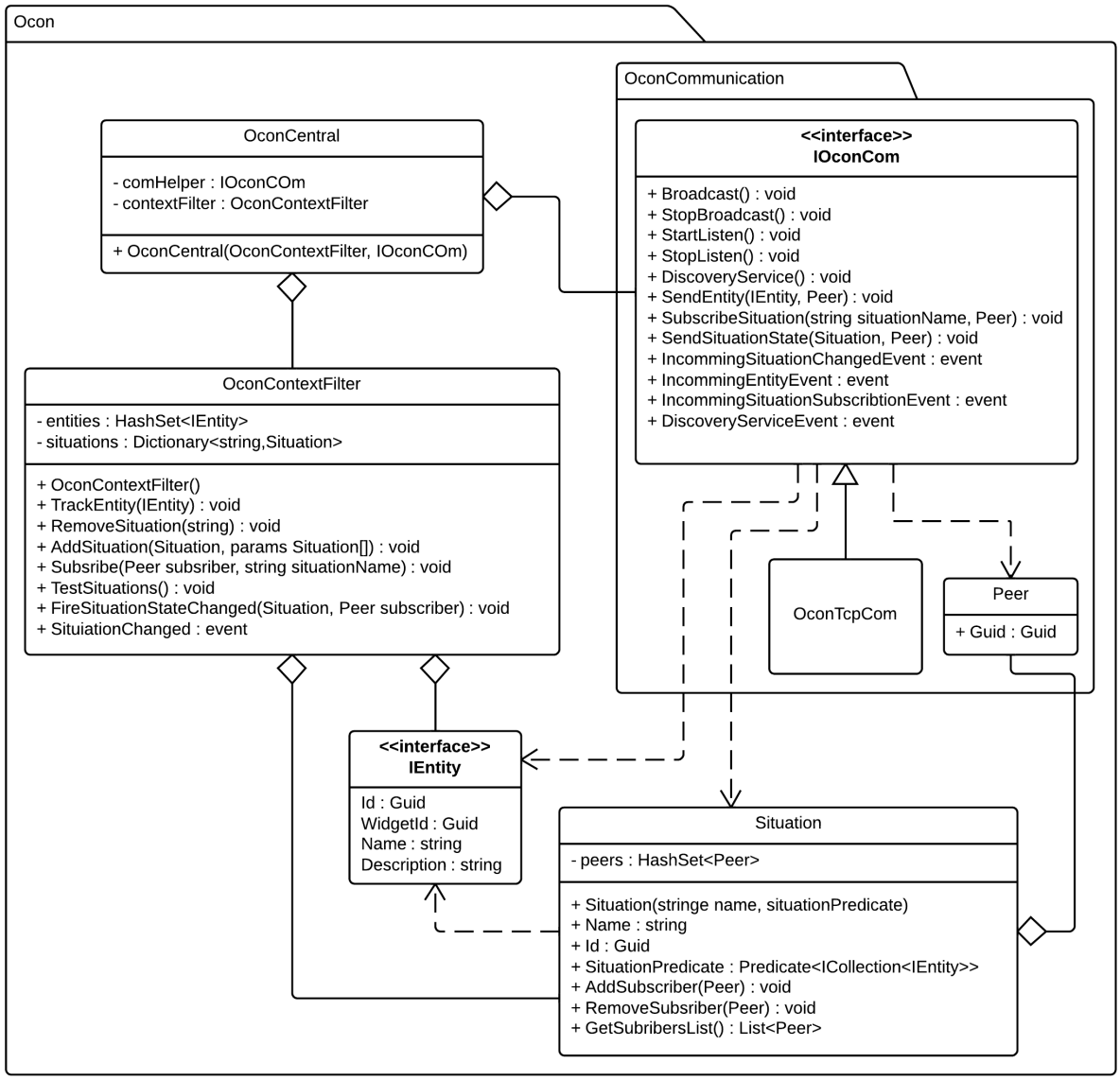


Figure 4.5: Class diagram showing the central and its usages

# 5 Proof of concept

> *"We can only evaluate the user experience afforded by the toolkit and its features by building applications that use it, and then evaluating them. While the toolkit itself can be evaluated on its technical criteria, the aspects of it that are designed to support a particular user experience can only be evaluated in the context of use and thus must be evaluated indirectly - through applications built with the toolkit."* [4]

In previous chapters we discussed the technical criteria of Ocon. As emphasized in the above quote we will in this chapter answer to our *Goal 2*, concerning usability, by implementing Ocon.

## 5.1 Motivation

Our proof of concept is a digital context-aware Scrumboard, and in this section we will briefly describe our motivation for this choice.

Scrum is a popular agile process for developing software. It focuses on rigid disciplines with timeboxing. Our motivation lies as part of the discipline of sprinting, and more precisely on the Scrumboard which has been adapted as a means to articulate the tasks in focus. This articulation is an important factor in teamwork which under the term articulation work describes the effort a team puts into communication[6, IV].

The classic physique of the Scrumboard is a whiteboard with post-its, but two reasons have been the driving force behind development of digital Scrumboards.

- The whiteboard is not digital, and there is labor in digitalizing its information. E.g. for a Scrummaster to calculate burndowns or velocity.
- Distributed Scrum teams have become more frequent with off-shoring and they need a digital solution for organizing tasks together.

Many tools have sprung up motivated by those factors[1]. These tools are not on par with the whiteboard when it comes to decreasing articulation work, because they lack the physical presence that the whiteboard has with a team. They do however solve the problems of distribution and digitalization.

[1] Confluence, Scrumwise, Team Foundation, Trello

**Motivation 1** *Popular digital Scrum tools are intended to be used by individuals. The lack of presence with the team increases articulation work compared to the whiteboard.*

Expanding on this is the motivation for context awareness as described in section 2.1 on page 4.

**Motivation 2** *Context awareness can improve the interaction between technology and humans*

Going forward in this chapter we will look to implement an idea of how these motivations could be satisfied by means of a digital Scrumboard and Ocon.

## 5.2 Situations and their informations

The Scrumboard's purpose as a boundary object is to display information relevant to the team[6, V]. Out of the information pool only some parts are interesting in a situation, and so we will use Context awareness to detect three situations in a Scrum environment and display relevant information on the Scrumboard accordingly.

- Overview. This situation is valid when no persons are in front of the board.

  *Information: Task overview*

- Closeup. This situation is valid when one person is detected in front of the board. Items on screen can be smaller and more detailed in this view than the overview.

  *Information: Task overview, burndown and calendar.*

- Standup. Envisioned for the Standup meeting, this situation is valid when 2 or more persons are detected. Items are medium sized and rather abstract.

  *Information: Task overview, burndown and calendar.*

Consult appendix A for screenshots of the scrumboard implementation with these ideas.

## 5.3 Implementation and environment

The situations described in section 5.2 are simple so to only require little context information - In this case whether persons are present. To encapsulate this we are extending on AbstractEntity since Ocon needs an IEntity type, and adding a present property. See figure 4.1.

For communication we will use OconTcpCom which depends on serialization between an Ocon-Widget and OconCentral. Therefore the Person implementation has to be known to both these projects.

The Context-aware Scrumboard consists of three parts: The KinectEntitySensor, The Centralization and the Scrumboard. These parts are distributed and communicate with the OconTcpCom implementation over LAN from each of their own hardware nodes. See deployment on figure 5.3.

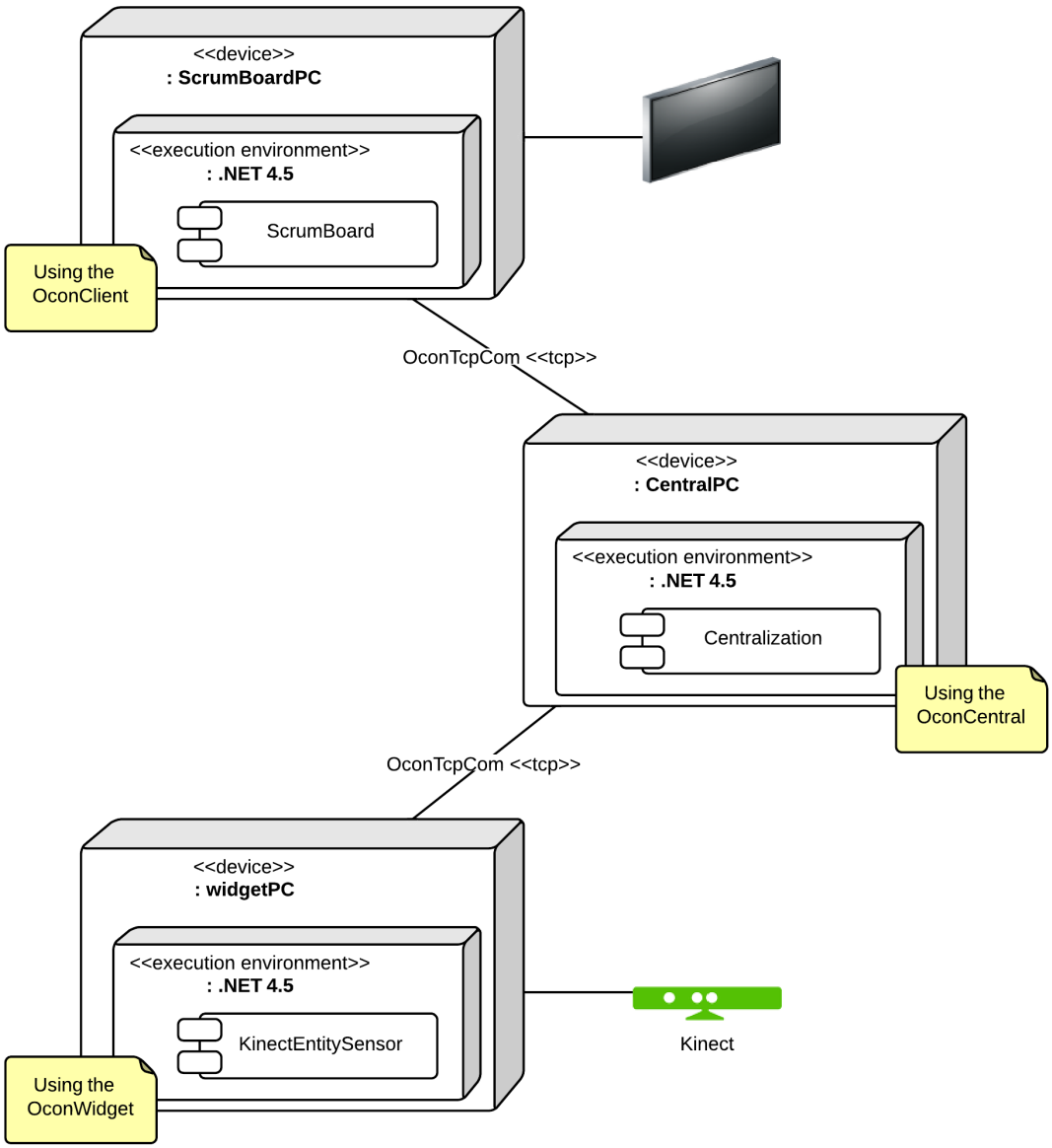


Figure 5.1: Proof of concept deployment

### 5.3.1 The KinectEntitySensor

The KinectEntitySensor's task is to gather Context Information and send it to the centralization. This part of the Context-aware Scrumboard implementation involves transferring information to the OconCentral, and it is for that purpose the OconWidget has been implemented. On figure 5.2 is a snippet of the implementation.

```
//Instantiate a logging instance
var log = Console.Out;
//for file logging: new StreamWriter("/file/path/here");

//Instantiate an IOconCom implementation
var com = new OconTcpCom(log);

//Instantiate widget
var widget = new OconWidget(com, log);

//Pass an entity to be added/updated at the central
var entity = new Person() { Name = "Mat", Present = true };
widget.Notify(entity);
```

Figure 5.2: Widget usage from TestWidget.Program

This code is mostly object instantiation followed by passing a hardcoded Person implementation to the OconWidget. Generally the job of this component is to transform captured context information into IEntity implementations and passing them to the OconWidget to be transfered to the OconCentral.

### 5.3.2 The centralization

The centralization has multiple tasks. First of all it's task is to receive Person objects from the KinectEntitySensor and coordinate them accordingly. Second of all it's task is to check registered situation predicates when entities are added or changed. And finally it's task is to notify subscribers when situation predicate states change.

```
//Instantiate a logging instance
var log = Console.Out;

//Instantiate an IOconCom implementation
var tcpCom = new OconTcpCom(log);

//Instantiate the context filter
var oconFilter = new OconContextFilter(log);

//Instantiate situations with names and predicates
var closeupSituation = new Situation("Closeup", e =>
    e.OfType<Person>().Count(p => p.Present) == 1);
var standupSituation = new Situation("Standup", e =>
    e.OfType<Person>().Count(p => p.Present) <= 2);

//Add the situations to the filter
oconFilter.AddSituation(closeupSituation, standupSituation);

//Instantiate the central
var central = new OconCentral(oconFilter, tcpCom, log);
```

Figure 5.3: Central usage from TestCentral.Program

### 5.3.3 The Scrumboard

The Scrumboard's task is to firstly register to situations on the Centralization and secondly bind and action with the OconClient which is fired when a subscribed situation's state has changed.

```
//Choose a logging instance if any
var log = Console.Out;

//Instantiate a network helper. Here passing the logging target
//alternatively instantiate as new TcpHelper(); if no logging is needed
var comHelper = new OconTcpCom(log);

//Instantiate the client with communication, log, and params of situation
    names strings
var oconClient = new OconClient(comHelper, log, StandupSituationString,
    CloseupSituationString);

//Subscribe a delegate to be run when a situation change event is fired
oconClient.SituationStateChangedEvent += (sender, args) =>
    UpdatePicture(args.SituationName, args.State);
```

Figure 5.4: Client usage from OconScrumBoard.MainViewModel

# 6 Reflection

**Predicate serialization** As mentioned in the implementation chapter, an obstacle occurred with serialization of predicates. This problem limited the usage of Ocon, but it was not possible to include the feature in this project. A framework for serializing the predicates as expression trees should be doable, but that is left for a later iteration of Ocon.

**Performance** While performance is not in scope it is still worth a few remarks, since it can have a big impact with our implementation approach. The predicate delegate is assigned by an implementor of Ocon and it is up to him what he does with the entity collection. Therefore in best case the growth is $O(situations)$. However an implementor can also break the system with blocking or long execution times.
Especially two points could improve performance when evaluating predicates:

- Parallel evaluation of predicates.
- Implementation of an algorithm to only check situations, depending on a given entity type, when an entity of that type is added or updated.

**Communication** The implementation of OconCommunication is a very large part of the framework. As per design goal 1 (see scope in section 1.2) our framework should be adaptable. Therefore OconCommunication is designed in a way so that developers can make their own implementation and dependency inject it into Ocon. The goal is achieved, but the solution could have been better. As it is now, developers have to change a very large part of the system. Reflecting on the implementation, a better design could have been achieved by decomposing IOconCom and OconTcpCom into following three components.

1. An interface for communication containing a send, listen, broadcast and discover method.
2. An interface for serialization containing serialize and deserialize.
3. A class using concrete implementations of the two interfaces above to achieve the functionality in the current OconTcpCom.

This would have resulted in a much lower coupled design making it much easier for developers to implement their own serialization or communication strategy.

Above reflections on communication, performance and serialization are obvious topics for future work to improve Ocon.

**Proof of concept** When implementing Ocon we experienced a framework that is easy to use and has the flexibility needed for our proof of concept. Especially the OconTcpCom implementation increased ease of use in our case, but imaging that another communication than tcp was needed, Ocon would function exactly the same given that this communication implementation was done correctly. Apart from that ease of use has been good in the simplicity of initializing the different Ocon components.

With that said one implementation by the developers of the framework only gives a limited view on its real-world usefulness

# 7 | Conclusion

In this section we will conclude on our project goals and thereby our project description contained by these goals.

**Goal 1** *Our aim is to implement a framework that is adaptable and easy to use.*

Ocon is adaptable in the way it allows for dependency injection but also transparent enough for ease of use. The OconTcpCom implementation being a big ease of use factor as seen in the proof of concept.

**Goal 2** *In trying to prove Goal 1, implement a stub Scrum board utilizing the framework*

A working stub Scrumboard utulizing Ocon has been implemented and used to prove Goal 1.

The implementations can be found at https://github.com/cholewa1992/Ocon

# Bibliography


[1] Mark Weiser *The Computer for thr 21st Century*, 1999.

[2] Dey, A. K., and Abowd, G.D. *Towards a better understanding of context and context-awareness. Workshop on the What, Who, Where, When and How Context-awareness, afflicted with the 2000 ACM Conference on Human Factors in Computer Systems*, 2000.

[3] Anind K. Dey and Alan Newberger *Support for Context-Aware Intelligibility and Control*, 2009

[4] W. Keith Edwards, Victoria Bellotti, Anind K. Dey, Mark W. Newman. *Stuck in the Middle: The Challenges of User-Centered Design and Evaluation for Infrastructure*, 2003.

[5] Anind K. Dey *Context-aware computing*, Chapter in Ubiquitous Computing Fundamentals by John Krumm at page 321-352, 2010.

[6] Lene Pries-Heje and Jan Pries-Heje *Why Scrum works*, 2011

# A Scrumboard screenshots

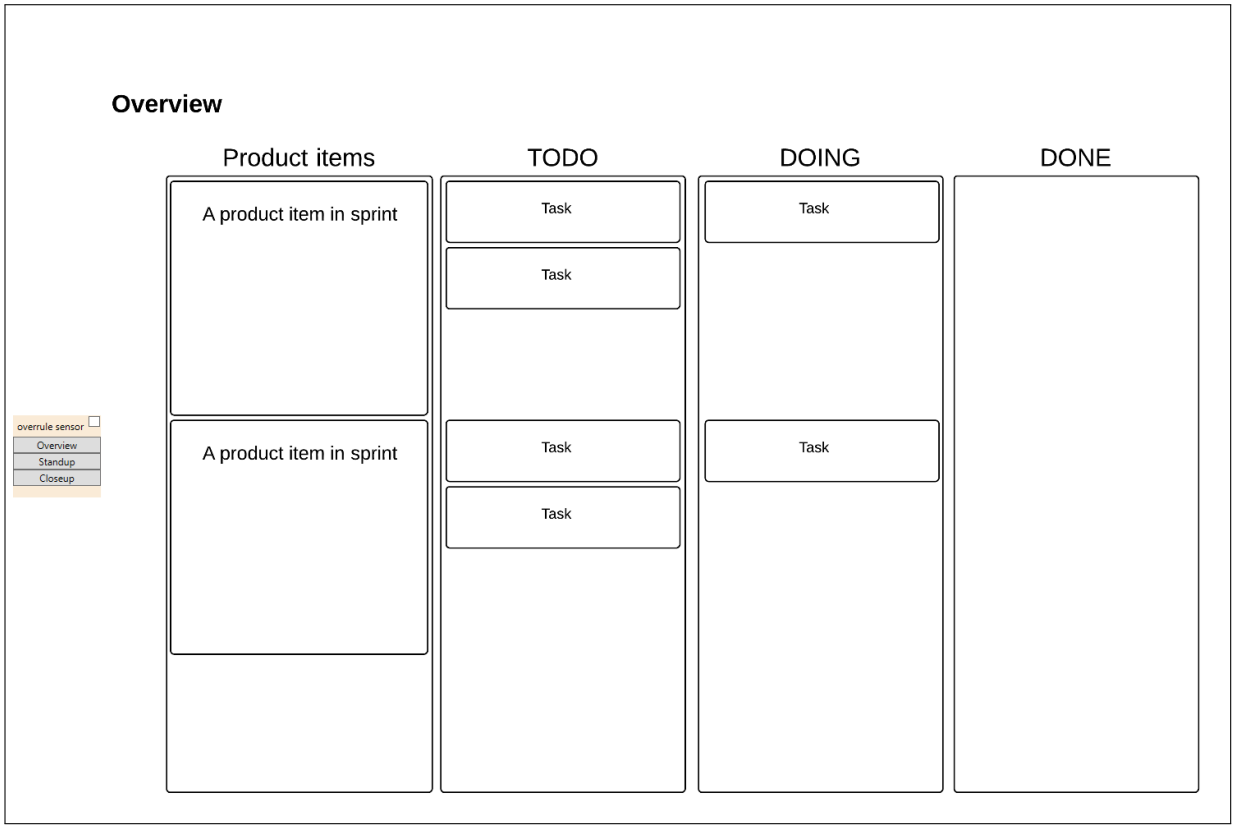


Figure A.1: Screenshot of the Scrumboard program in Overview

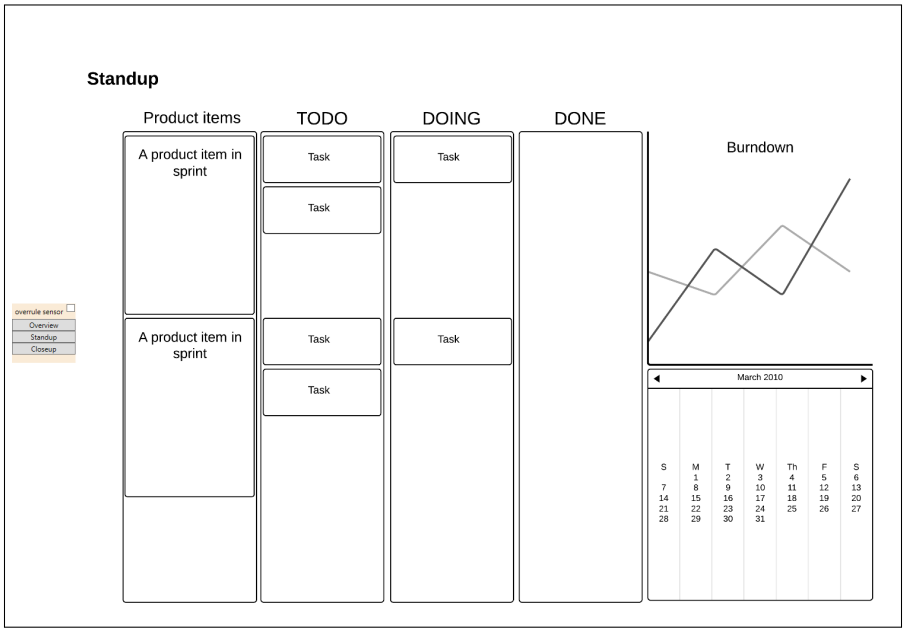


Figure A.2: Screenshot of the Scrumboard program in Standup

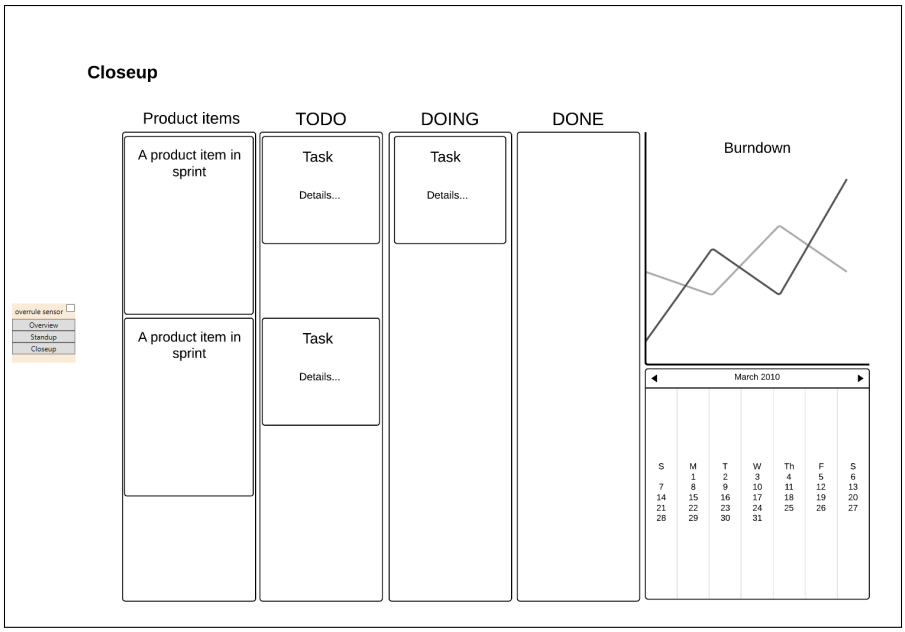


Figure A.3: Screenshot of the Scrumboard program in Closeup. This is a more detailed view than the Standup