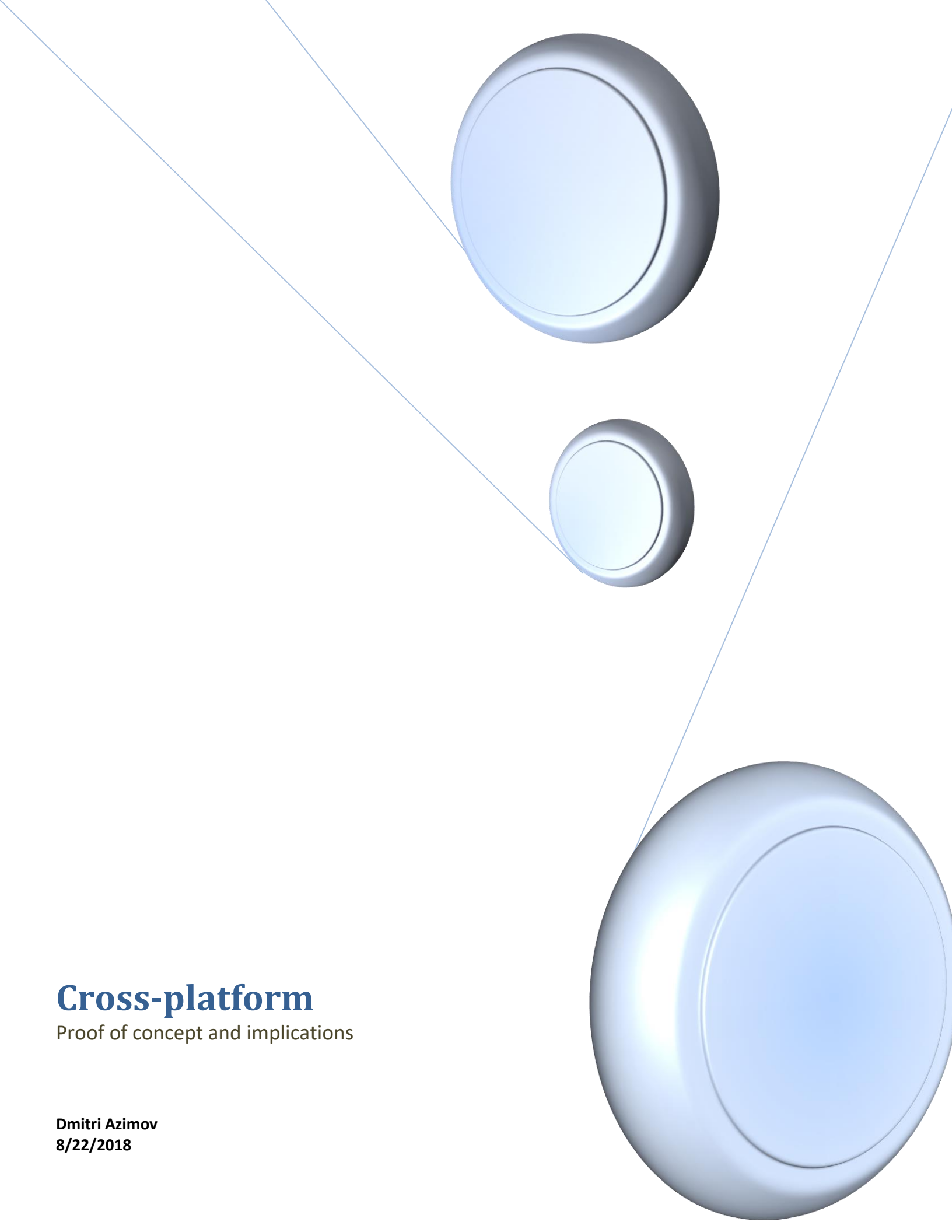


Cross-platform

Proof of concept and implications

Dmitri Azimov
8/22/2018

A cross-platform feature can drastically change the way a user interacts with the game. Having the option to use a mobile device instead of a generic controller puts the player a more versatile way of communicating with the game environment.

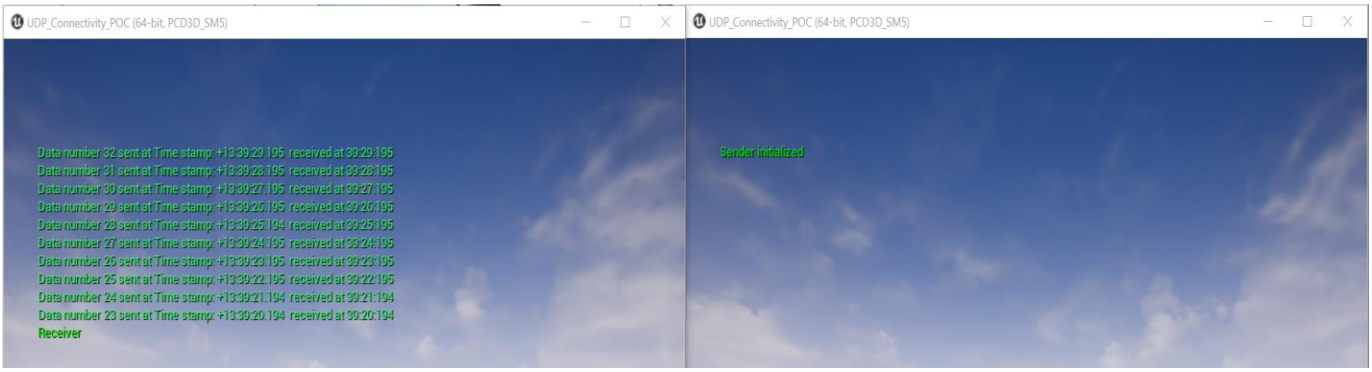
The player would have a visual interface displaying much of the needed information, thus minimizing the amount of UI present on the main screen, rendering the game less cluttered, and therefore more immersive. The phone can go beyond being a one way input tool; it could heighten the player experience with input from the game. The controller could react to game events by, for example, vibrating or flashing when the character gets hurt. The player could even use his phone as an interface to customize his character without disturbing the game for other players. The possibilities are ever extending.

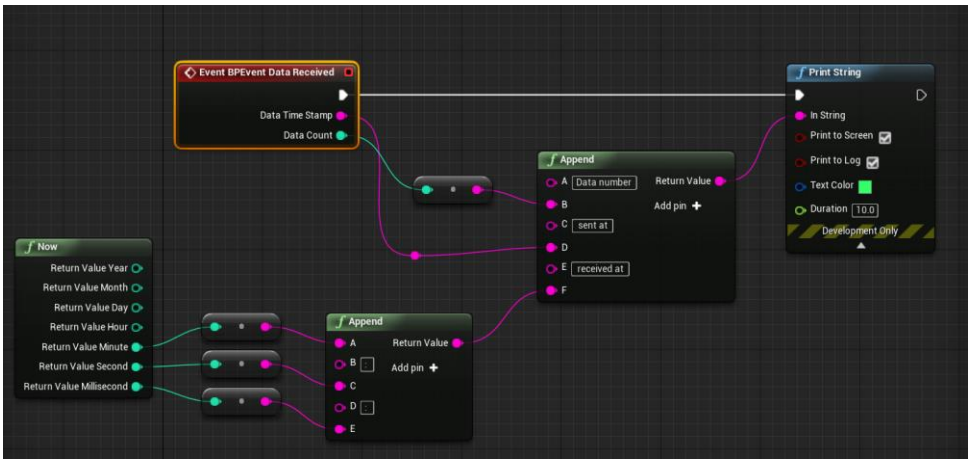
In addition, the possibility of having an extremely common device replace the classic controller adds a layer of convenience that is bound to attract more users to the product.

For the technical side, below we have two instances of Unreal communicating with each other over UDP. This serves as a proof of concept to show that we can establish communication between devices on a local network without any significant lag or delay.

On the right, we have a sender instance. It is sending serialized information that is then picked up almost instantly by the receiver on the right. That information could be any serializable data type and therefore offers more than enough flexibility for us to add to the player experience.

UDP was used for demonstration purposes, but there is nothing bounding us to this protocol. In fact, it would be a better idea to use transmission control to make sure the listening instance receives the packets in order. The steps required for the additional protection won't necessarily affect transmission performance. Since all communication is done on a local network, the delay is negligible and won't be perceived by the user.





Receiver

The Unreal instance on the left has a receiver actor that opens a socket with the same address as the sender's endpoint. It continuously listens for packets and prints out the data to the screen upon receiving it.

```
void AReceiver::Receive(const FArrayReaderPtr & array_reader, const FIPv4Endpoint & end_point)
{
    FTransferableData data;
    *array_reader << data;

    BPEvent_DataReceived(data);
}

bool AReceiver::StartReceiver(const FString & socket_name, const FString & ip, const int32 port)
{
    FIPv4Address address;
    FIPv4Address::Parse(ip, address);

    FIPv4Endpoint Endpoint(address, port);

    int32 buffer_size = 2*1024*1024;

    listen_socket = FUdpSocketBuilder(*socket_name).AsNonBlocking().AsReusable().BoundToEndpoint(Endpoint).WithReceiveBufferSize(buffer_size);

    FTimespan thread_wait_time = FTimespan::FromMilliseconds(100);
    receiver = new FUdpSocketReceiver(listen_socket, thread_wait_time, TEXT("UDP RECEIVER"));
    receiver->OnDataReceived().BindUObject(this, &AReceiver::Receive);
    receiver->Start();

    return true;
}
```

```cpp
void ASender::SendString()
{
    if (!sender_socket)
    {
        if (GEngine)
            GEngine->AddOnScreenDebugMessage(-1, 0.1f, FColor::Green, TEXT("bad socket"));
    }

    int32 bytes_sent = 0;

    FTransferableData data;
    data.count = ++count;
    data.time_stamp = "Time stamp: " + (FDateTime::Now().GetTimeOfDay()).ToString() + " ";

    FArrayWriter writer;

    writer << data;

    sender_socket->SendTo(writer.GetData(), writer.Num(), bytes_sent, *remote_address);

    if (bytes_sent <= 0)
    {
        if (GEngine)
            GEngine->AddOnScreenDebugMessage(-1, 0.1f, FColor::Yellow, TEXT("nothing sent"));
    }
}

bool ASender::StartUDPSender(const FString & socket_name, const FString& ip, const int32 port, bool UDP)
{
    remote_address = ISocketSubsystem::Get(PLATFORM_SOCKETSUBSYSTEM)->CreateInternetAddr();

    bool ip_is_valid;
    remote_address->SetIp(*ip, ip_is_valid);
    remote_address->SetPort(port);

    if (!ip_is_valid)
    {
        if (GEngine)
            GEngine->AddOnScreenDebugMessage(-1, 10000.0f, FColor::Red, TEXT("Invalid sender IP"));
        return false;
    }

    sender_socket = FUdpSocketBuilder(*socket_name).AsReusable().WithBroadcast();

    check(sender_socket->GetSocketType() == SOCKTYPE_Datagram);

    int32 send_size = 2*1024*1024;
    sender_socket->SetSendBufferSize(send_size, send_size);
    sender_socket->SetReceiveBufferSize(send_size, send_size);

    if (GEngine)
        GEngine->AddOnScreenDebugMessage(-1, 10000.0f, FColor::Green, TEXT("Sender initialized"));

    GetWorldTimerManager().SetTimer(send_timer, this, &ASender::SendString, 1.0f, true, 1.0f);

    return true;
}
```

Sender

The Unreal instance on the right contains a sender actor. It creates a socket with a defined address, and then uses that endpoint to periodically send information. Here the information is sent every second, but in reality it would do so whenever user input is detected.