Ambient Flow: A Visual Approach for Remixing the Internet of Things

Darren Carlson*, Matthias Mögerle†, Max Pagel*, Shivam Verma*, and David S. Rosenblum*
*Felicitous Computing Institute, National University of Singapore
{carlson, pagel, shivam, david}@comp.nus.edu.sg
†Institute for Visualization and Interactive Systems, University of Stuttgart
matthias.moegerle@studi.informatik.uni-stuttgart.de

***Abstract*—The number of networked "smart devices" available in everyday environments is rapidly increasing; however, most adopt mutually incompatible networks, protocols, and application programming interfaces. In previous work, we introduced a variety of adaptive middleware techniques that enables a user's commodity mobile device (e.g., a smartphone) to serve as an adaptive gateway between mutually incompatible devices – providing service adaptation and protocol translation services through plug-ins that can be installed on-the-fly. In this paper, we present a complementary set of novel smart space design tools, which enable non-programmers to visually "remix" their ambient environments in new, playful and potentially unforeseen ways using an intuitive flow-graph model. Visual designs can be sent over the network to the user's mobile device, which instantiates them during runtime. This paper presents a detailed overview of the approach, introduces our fully functional prototype, and presents a user study that provides encouraging first results.**

I. Introduction

As the Internet of Things (IoT) rapidly becomes a reality, a flood of new consumer smart devices is beginning to enter the market, permeating many environments. We are now able to easily connect to these devices using a variety of different network technologies, such as Bluetooth, Bluetooth LE, WI-FI, Near-Field-Communication (NFC), ANT+, 6LOWPAN and more. While such capabilities have ostensibly made it easy to discover and communicate with these new devices, no universally adopted application-layer protocols or interaction patterns have yet emerged. As a consequence, the (IoT) is becoming a highly fragmented landscape of vendor specific protocols, where seamless smart device interactions *across* (IoT) system boundaries is complex, error prone, and increasingly rare.

Although a variety of "smart gateway" devices (e.g., home hubs) endeavor to act as a common connection point for nearby connected devices, these devices must be physically installed and configured for each target environment, limiting their accessibility. Moreover, smart gateway implementations typically limit the supported devices and protocols to those known a priori. In the rapidly emerging IoT, these limitations become highly problematic, since the number of required smart gateway servers will increase rapidly, as will the number of potentially unknown smart devices – leaving potentially useful and powerful *cross-vendor* combinations of smart device functionalities hidden.

In previous work [1], we described how a commodity mobile device can be transformed into an adaptive smart gateway that is carried by the user, providing service adaptation and protocol translation services through plug-ins that can be installed on-the-fly. The foundation of this approach, called Ambient Dynamix [2] (Dynamix), is our plug-and-play mobile middleware framework that is capable of loading and instantiating a variety of plug-ins into the user's device at runtime, in order to provide dynamically extensible context sensing and remote control services. Related, we also developed the *Ambient Control* library (AC Library) for Dynamix, which aims to unify common control patterns between smart devices, enabling the ad-hoc discovery, selection and integration of smart devices in highly heterogeneous environments [3].

To demonstrate the interaction possibilities provided by the AC Library, we introduced the *Tap To Interact* workflow [4], which allows users without coding experience to "wire" smart devices together by tapping affixed Near Field Communications (NFC) tags using their Dynamix-enabled device. This capability facilitates easy, fun and sometimes unexpected ways to interact with encountered connected devices. Although *Tap To Interact* is useful in small-scale scenarios (e.g., connecting two or three well-known smart devices), the approach can become quickly insufficient for configuring larger smart space scenarios. In particular, *Tap To Interact* lacks the rich visual feedback necessary for creating complex smart device orchestrations; requires that participating smart devices be known a-priori and labeled with NFC tags; and does not support the saving or sharing of developed smart space configurations.

Based on these limitations, this paper presents a novel visual approach for "remixing" ambient environments using a Web-based toolset that can be accessed from the user's desktop computer. This approach, called Ambient Flow, leverages information from the user's mobile Dynamix instance to render discovered connected devices as block diagrams in a Web canvas that can be "wired" together using an intuitive flow graph model. Ambient Flow executes completed graphs by automating the installation of required plug-ins and control logic into the user's Dynamix instance during runtime. The proposed approach is intended to allow non-programmers to actively discover, connect and share smart space configurations in a fun and playful manner. As such, it focuses on lowering the complexity inherent to smart space orchestration through efficient user guidance and automated configuration.

This paper is structured as follows. Section II provides background on the Ambient Dynamix framework and the Ambient Control library, which form the foundation for this current work. Section III introduces the Ambient Flow approach, including its key design principles, software architecture and smart space configuration model. Section IV describes our fully operational prototype system, which is used to validate the Ambient Flow approach. Section V presents a preliminary user evaluation that investigates how end-users can utilize the prototype to configure their environments. Section VI describes related work. Section VII concludes the paper with a discussion of our contributions and an outlook on future work.

II. Background

The number of networked smart devices available in everyday environments is rapidly increasing; however, many current devices adopt mutually incompatible networks, protocols, and application programming interfaces (APIs). For example, devices like the Sphero Robotic ball and Parrot AR Drone helicopter both provide dedicated controller apps, but remain inherently incompatible. The Sphero supports sensor data streaming of its inertial measurement unit (IMU) over a Bluetooth connection, which can be used to determine the current orientation of the robot using a proprietary API. The AR Drone supports data connectivity over WiFi and accepts flight control commands that can be used to remotely pilot the drone. A grasped Sphero device could theoretically be used as an intuitive flight controller for the drone (by feeding its streaming IMU data into the drone's flight control API); however, such interactions are not inherently supported by the devices.

A. Ambient Dynamix

To address the network, protocol and API heterogeneity challenges introduced above, we developed *Ambient Dynamix* (Dynamix for short) [2], a plug-and-play middleware framework that enables mobile apps and Web apps [5] to sense the user's context (e.g., location, identity, activity) and perform fluid smart device interactions through plug-ins that can be dynamically installed into the user's Android-based mobile device (e.g., smartphone or tablet) on-demand. Dynamix runs as lightweight background service, leveraging the user's mobile device itself as a sensing, processing and communications platform. Dynamix comes with a growing collection of ready-made plug-ins and provides open software development kits (SDKs) and a scalable repository architecture, which enable 3rd party developers to quickly create and share new plug-in types with the community. An overview of the Dynamix Framework is shown in Figure 1.

As shown above, a Dynamix Service is situated between a device's local hardware and (potentially many) Dynamix apps. Apps communicate with a Dynamix Service through easy-to-use application programming interfaces (APIs), including a Facade API (for requesting and controlling context support) and an Event API (for receiving framework notifications and context events). Dynamix automatically discovers, downloads and installs the plug-ins needed for a given sensing or control task. Plug-ins can be manually installed into a Dynamix Service or deployed automatically in the background in response to app requests (from a configurable set of public or private repositories). Plug-ins are automatically deployed along with all their dependencies (e.g., linked code libraries, resources, assets), which allows Dynamix-based apps to easily install complex context support using only a few lines of code. Plug-ins are packaged and deployed as OSGi bundles, which are

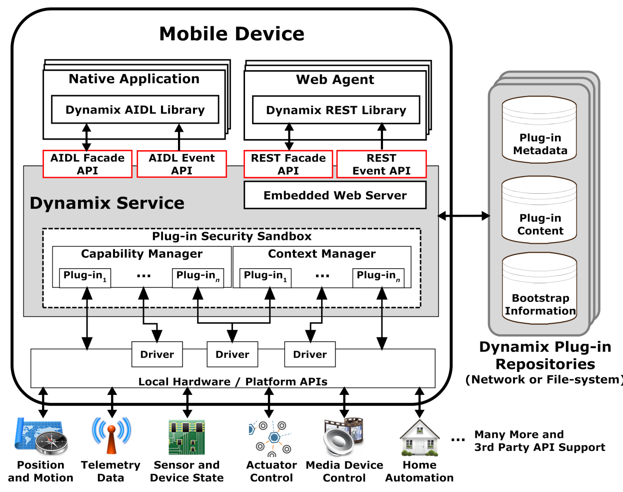

Fig. 1. Overview of the Dynamix Framework

Java Archive (JAR) files with additional metadata. When the user changes environments, new or updated plug-ins can be deployed to the device at runtime, without the need to restart the framework.

Dynamix includes comprehensive inter-plug-in communication capabilities that enables plug-ins running within a Dynamix instance to send messages to each other and consume services provided by other plug-ins (e.g., register for context sensing or device control services). This opens up the possibility of dynamically installing service adaptation and/or protocol translation plug-ins, which enables a Dynamix device to serve as a mediator between mutually incompatible smart devices situated in the user's environment. These capabilities provide the foundation of the Ambient Control approach described next.

B. The Ambient Control library

To enable dynamic "remixing" of encountered IoT resources, we developed the *Ambient Control* (AC) library, which aims to unify common control patterns between smart devices, enabling the ad-hoc discovery, selection and integration of smart devices in highly heterogeneous environments. The AC library loads into a Dynamix instance and enables the user's mobile device to serve as a mediator between mutually incompatible smart devices. The AC library provides extensible mechanisms for describing, discovering, and combining the command messages and associated data provided by a wide variety of smart devices through Dynamix plug-ins that can be loaded at runtime. This capability facilitates easy, fun and sometimes unexpected ways to interact with connected devices.

To enable a smart device to be automatically wired by the AC library, a Dynamix developer first creates a plug-in that exposes the device's features using standardized Dynamix commands. Next, the developer publishes a plug-in profile that describes the possible commands that the underlying device can produce and consume. (Note that it is also possible to specify the minimum set of control commands needed to control a device, and which controls are optional.) The AC library supports 1-to-n connections that involve 1 receiver and any number of controllers. We call such a configuration a control graph, since it consists of nodes (plug-ins) and directed control edges between nodes, over which certain types of controls are exchanged. Multiple control graphs can operate simultaneously within a single Dynamix instance. The AC Library includes a Smart Wiring feature that optimally matches the inputs and outputs of the plug-ins in a given control graph according to priority values. The AC Library coordinates requested control graphs by managing required plug-in installations via Dynamix, handling the setup handshake process between plug-ins and managing full duplex communication channels between controllers and a receiver. Detailed information about this approach can be found in [3].

A single Dynamix instance equipped with the AC library can run multiple control graphs simultaneously in order to support complex interactions among multiple devices. We call such a set of one or more Control Graphs a "control scenario". Figure 2 shows an example Control Graph configuration connecting a Sphero robot as a controller to a Parrot arDrone receiver. This configuration also includes a translator that mediates between incompatible control message requirements.

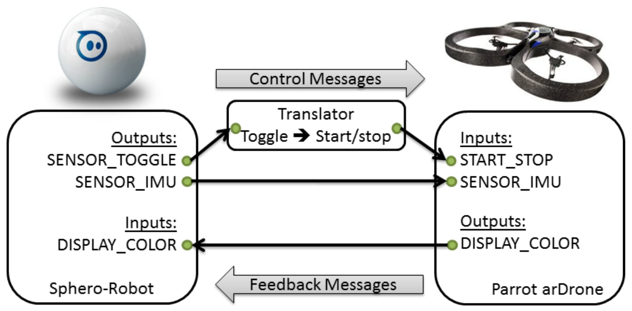

Fig. 2. Sample Control Graph configuration

In order to facilitate interactions between smart objects, the AC library defines a set of control commands that define well-known interaction semantics and associated data. These commands are defined as string constants with values like "MOVEMENT_FORWARD" or "PLAYBACK_PLAY", and are used to identify the types of command messages that may be exchanged between linked plug-ins. Each command message may hold additional parameters, depending on the type of command (e.g., a reference to the video file to be played in the case of a "DISPLAY_VIDEO" message). We also defined several sensor message types that can be used to transmit raw sensor data from one plug-in to another in a standardized way. These messages are also identified by string constants like "SENSOR_IMU" (e.g., for data from an orientation sensor). A message of this type would contain the sensed orientation as pitch, yaw and roll values within standardized ranges. It is the plug-in developer's responsibility to transform the raw values delivered by a plug-in into the range defined by the AC library documentation. For example, for "SENSOR_IMU" control commands, we define that the pitch value must be in the interval [-180,180], with 0 in a horizontal normal orientation and -180 if horizontal but upside down.

Control commands can be associated with a Dynamix plug-in through a plug-in control profile that describes the commands a given Dynamix plug-in can emit (output) and consume (input), along with the associated data types that can be exchanged. To reflect different sets of commands that can be used to control a plug-in, it is possible to create several input profiles per description with different priorities. Plug-in control profiles are stored inside a Dynamix web service, which offers a REST interface for accessing the profiles based on a plug-in's id, as well as the ability to perform queries for plug-ins that support or require certain command types. This allows developers to integrate new plug-ins with the AC library any time without recompiling or restarting the Dynamix service, by leveraging the dynamic download and installation capabilities of Dynamix [2].

III. Ambient Flow

The approaches described in the last section are able to solve many of the interoperability challenges inherent to heterogeneous smart environments. Through a provided smart space configuration control graph, Ambient Control is able to automatically configures its Dynamix instance to serve as a smart gateway between the specified networked devices. Although manual control graph creation works well in experimental scenarios, configuring real-world environments requires addressing multiple interaction patterns and a larger number of smart devices, which can quickly become cumbersome and complex when writing control graph configurations by hand. In this section, we explore how the creation of such configurations might be simplified, according to the 3 design principles outlined below.

A. Key design principles

1) *Simplicity:* Users should be shielded of the low-level technologies underlying the IoT, such as communication protocols, data-types, and idiosyncratic implementations of service interfaces. Users should be free to easily mix and match rich sources of contextual data and smart device capabilities to create smart space scenarios that fits their needs.
2) *Immediacy:* Users should be guided and supported in the creation of working IoT scenarios, with an emphasis on immediate feedback and minimizing frustrating errors. It should be possible to immediately test new scenario configurations and obtain relevant real world feedback on their viability.
3) *Playfulness:* It should be possible for users to explore new and creative interaction scenarios in a playful way that encourages experimentation with IoT resources.

B. Architecture

Based on the design principles outlined above, we devised a smart space configuration approach, called Ambient Flow, which enables non-programmers to easily create and adjust control graphs visually, and then load them into a paired Dynamix-based device for realization. Control graphs can be loaded into a Dynamix instance either from the network or from other sources (e.g., from another Dynamix plug-in or app). The aim of Ambient Flow is to simplify the configuration and orchestration of smart spaces, while encouraging users to explore possibilities in their environments in order to come up with their own creative use-cases. The next sections described the Ambient Flow approach by addressing each of the key usability principles outlined above.

To address the simplicity and immediacy principles, we explored representing IoT scenarios using a flow-based programming model. Flow-based programing [6] is a visualization technique that allows users without coding experience to understand, create and alter programs by manipulating graphical blocks that represent program components or functionalities. These blocks expose available parameters and interaction possibilities, such as inputs and outputs. Completed control graphs can be collapsed into a single subgraph to allow for a more tidy representation that can be included as a single block in more complex scenarios (e.g., configurations that combine components comprised of large numbers of low-level input/output connections). Global inputs and outputs for a complete subgraph can be assigned, which enables interactions between graphs or groups of smart objects. By configuring individual blocks, and connecting the inputs and outputs of various blocks together, a user can create a fully functional program that solves a given task.

We applied the flow-based programming approach to the IoT interaction capabilities provided by Ambient Control. Towards this end, we created a Web-based ”Flow Designer” that renders smart devices and their available inputs and outputs as graphical elements within a Web-based canvas. Using the Flow Designer, users can drag smart device blocks to the canvas and then connect them together to create complex graphs that represent the intended configurations of a smart space. Completed configurations can be deployed directly to a Dynamix-based device or shared with other users through an online repository called the Ambient Flow Server. An overview of the Ambient Flow network architecture is shown in Figure 4.

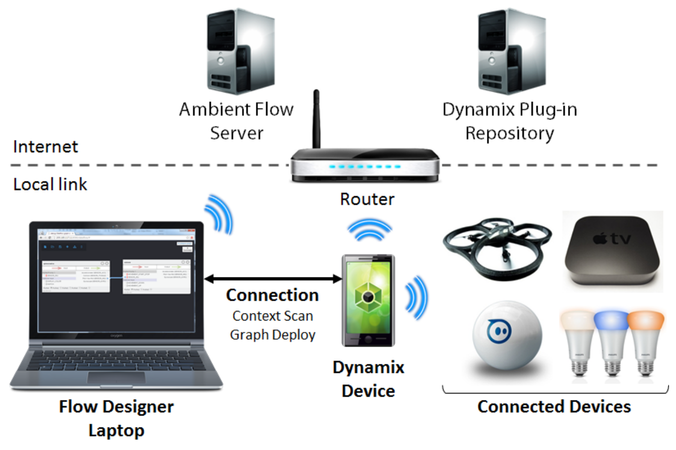

Fig. 4. Ambient Flow network architecture

To address the principle of playfulness, the Flow Designer’s interface provides constant feedback during the creation process by assisting the user when selecting possible connection points between devices. For example, if the user begins dragging an output of a smart device block, only compatible inputs on potential target smart devices are shown as valid (incompatible inputs are grayed out). If the user attempts to connect incompatible outputs and inputs, required translators are automatically inserted if available, without the need for additional user interaction. This prevents the user from configuring faulty interaction patterns that wouldn’t work in reality.

Figure 3 depicts a completed control graph connecting a Sphero robot ball and a Parrot ARDrone within the Flow Designer. As the Sphero is able to produce orientation data from its inertial measurement unit (SENSOR_IMU), this output can be directly connected to the IMU input of the drone, controlling the drone’s flight direction. The second input required to control the drone is a command telling it to take off or land (MOVEMENT_START_STOP). In this example, this command is provided by connecting the collision sensor of the Sphero in the form of toggle events, which are translated into the necessary MOVEMENT_START_STOP commands using a translator, as shown in Figure 3. Through this configuration the user can start or land the drone by tapping the Sphero ball and thus setting of the collision sensor. Once flying, the Ambient Control plug-ins match the drone’s orientation to the orientation of the sphere in the users hand, creating an intuitive flight control system based on the user’s hand gestures.

To address the immediacy challenge, the Flow Designer running in a desktop browser can be remotely paired with the user’s Dynamix instance running on a smart phone. Each Dynamix instance exposes its APIs using an inbuilt web server that provides a fully featured REST interface to remote devices. This connection allows the Flow Designer to scan the environment for available smart devices and quickly deploy completed control graphs for testing.

To support remote pairing, Dynamix offers an intuitive service called ”Scan to Interact”, which uses its barcode plug-in to read pairing information from a 2D barcode that can be generated and displayed in the Flow Designer’s interface. The Dynamix device is used to scan the barcode and read the pairing information. This information includes metadata including the Web app’s name, a pairing code, and a shared secret that is used to sign and verify API calls. These credentials are generated on-the-fly by the Flow Designer and are only shared optically with Dynamix. Once the pairing barcode is read by Dynamix, it publishes its server IP address and port using a public Dynamix Web service. The Flow Designer monitors the pairing Web service, and automatically binds to the Dynamix device once the pairing code and IP address are published. Details of this remote pairing approach will be presented in an upcoming paper.

Once the Dynamix instance on the phone is bound with the Ambient Flow Web interface, the control graph can be sent to the AC Library running within the Dynamix instance on the smartphone. The AC Library parses the incoming control graph, and then instructs its Dynamix instance to install the plug-ins required to support the smart devices involved in the configuration. Once plug-in support is available, the AC Library will then initialize messaging between the various plug-ins as a means of realizing the control graph.

IV. PROTOTYPE IMPLEMENTATION

To validate the Ambient Flow approach presented in the last section, we created a fully operational prototype that includes the Flow Designer, Ambient Flow Server (AFS) and the aforementioned Dynamix remote pairing capability

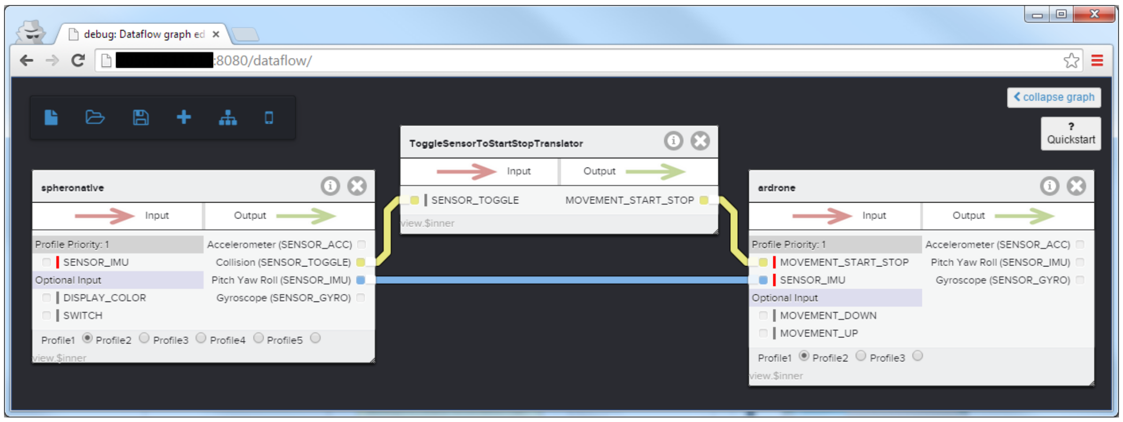

Fig. 3. The Flow Designer displaying a simple configuration that enables a Sphero robot to be used as a controller for a Parrot ARDrone

(see Figure 4). The AFS hosts a database that contains the available Ambient Control control profiles, and the various control graphs published by users. Recall that control profiles provide a generic way of describing the high-level capabilities provided by smart devices (e.g., inputs, outputs, and associated data types), as provided by a Dynamix plug-ins. The AFS also hosts the Flow Designer, which is implemented as an HTML/CSS/JavaScript framework that executes within a Web browser.

The Flow Designer was implemented by extending the open-source Meemoo framework [7], which provides high-level visual programming features such as block rendering, wiring support, subgraphs, etc. The Flow Designer utilizes the REST interface of the AFS to obtain the metadata of available control profiles. These are rendered as smart device blocks that can be placed on the canvas. The input and output data-types for each smart device block are extracted from the retrieved control profile metadata. Blocks can be placed on the canvas by clicking the ”Add Devices” button in the Flow Designer’s interface, which displays both available and unavailable (i.e., offline or not present) devices in a list. Clicking a device in the list places it on the canvas, where it can be connected with other smart devices.

The Flow Designer (running on a desktop Web browser) supports remote pairing with a remote Dynamix instance (running on an Android mobile device) when situated in the same local network. The pairing process operates using the previously mentioned out-of-band credential exchange (a pairing token shared optically via a barcode). Once the Flow Designer is paired with a Dynamix instance, the Flow Designer’s canvas can be populated with smart device blocks that contain ”live data” from the user’s environment, such as available devices with corresponding id information and state. Connections between blocks are made by dragging ”wires” between device outputs and inputs. As described earlier, the interface guides users when creating connections by graying out invalid targets (see Figure 5), and by adding translators as required. Completed control graphs can be sent over the network from the Flow Designer to the paired Dynamix device for testing. The Ambient Control library running within the Dynamix device receives incoming control graphs (as XML), parses the configuration, uses the Dynamix instance to install specified plug-ins, and then sets up the plug-in intercommunication

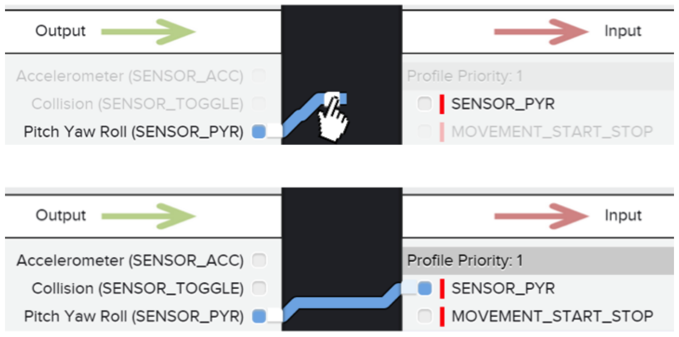
Output
Input
Accelerometer (SENSOR_ACC)
Collision (SENSOR_TOGGLE)
Pitch Yaw Roll (SENSOR_PYR)
Profile Priority: 1
SENSOR_PYR
MOVEMENT_START_STOP
Output
Input
Accelerometer (SENSOR_ACC)
Collision (SENSOR_TOGGLE)
Pitch Yaw Roll (SENSOR_PYR)
Profile Priority: 1
SENSOR_PYR
MOVEMENT_START_STOP

Fig. 5. Example of user guidance during graph editing process

channels necessary to render the graph. As completed control graphs contain live data from the environment, the underlying plug-ins are able to properly connect with specified devices and provide requested protocol translation services. During runtime, completed designs can be utilized by Dynamix-based devices alone, without requiring the Ambient Flow tooling.

In previous work, we demonstrated that complex resources, such as support libraries, native code, and protocol stacks, can be successfully bundled and deployed within Dynamix plug-ins [8]. For the Ambient Flow prototype presented in this section, we created several Dynamix plug-ins that use these capabilities to support protocol translation (e.g., via runtime-deployed native protocol libraries) and dynamic integration via Ambient Control. Towards this end, each plug-in provides high-level descriptions of their capabilities (i.e., control profiles), which enables automatic ”wiring” of their control logic to each other according to the AC Library running within the Dynamix instance. The plug-ins developed for this prototype implementation support network-based lighting (Phillips Hue and LIFX); the AR Drone Helicopter; the Sphero Robotic Ball; the Myo Armband; various Wemo switches and motion detectors; and networked media players, such as Apple TVs and UPnP players. Additional plug-ins are in development.

V. User Study

To validate the usability of the initial Ambient Flow prototype, we conducted a preliminary user study with a total of 10 participants (all were non-programmers). The user study included 6 male and 4 female participates, ranging in age from 16 to 50 years old. The study was conducted to investigate how well users without programming experience were able to create IoT interaction scenarios using the Flow Designer. During the beginning of the study, users were provided with a short, Web-based tutorial that provided a guided tour through the Ambient Flow web interface. The tutorial instructed users how to perform basic tasks using the Flow Designer, such as placing and connecting smart-device blocks. The tutorial took less than 5 minutes to complete.

After finishing the tutorial, users were then given a smart space configuration task that required them to create and verify a new control graph using the Flow Designer. The task required that users connect a Sphero robot to a Hue network light in such a away that the orientation of the robot controlled the color of the Hue light. Users were instructed to use the Flow Designer to add appropriate block components to the canvas, connect the blocks to achieve the specified task, and then deploy the completed graph to a paired Dynamix-based smartphone for testing.

Although the Sphero's role as a lighting controller is not inherently obvious, the Flow Designer's user guidance system helped participants make this type of unexpected connection by highlighting available wiring points and adding translator support as needed. As shown in Figure 6, by connecting the Sphero's SENSOR_IMU output data pin to the DISPLAY_COLOR input pin on the Hue light, the Flow Designer automatically installs a translator that maps orientation data to color values, allowing users to creatively try new interaction possibilities.

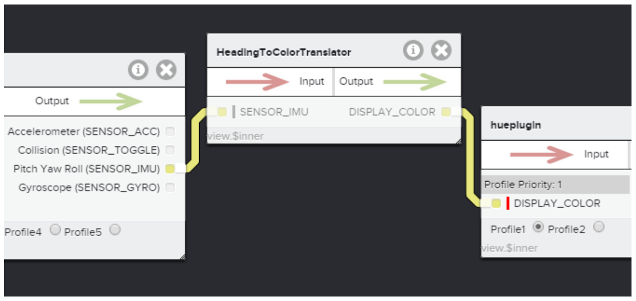

Fig. 6. The completed user study control graph

Users were also instructed to utilize the previously described Dynamix remote pairing feature to deploy and test their control graph using real world hardware. To accomplish this, users clicked the "Connect to Dynamix" button in the Flow Designer, which displayed a barcode that could be scanned by the user's Dynamix-based device to complete the pairing process. After pairing, the Flow Designer automatically deployed the completed control graph to the Ambient Control library running within the Dynamix instance on the user's mobile device. As previously discussed, the AC Library automatically parses incoming control graphs, uses its Dynamix instance to install and instantiate requested plug-ins, sets up control messaging, and manages resulting data flows during runtime. As shown in Figure 7, users were able to test deployed control graphs by rotating the Sphero device to control the color of the Hue light.

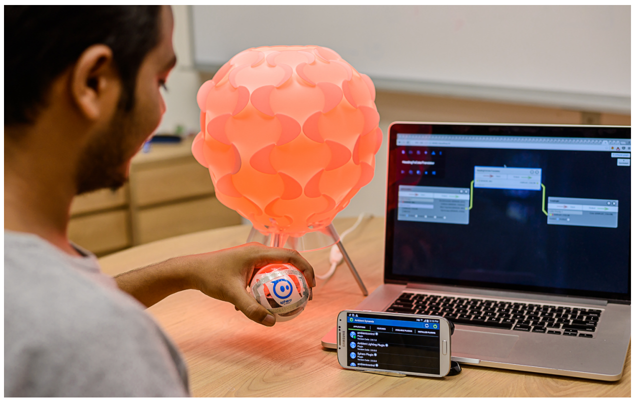

Fig. 7. A user study participant completes the task of connecting a Sphero robot ball to a Hue-based smart lamp.

The results of this preliminary user study are encouraging. All users were able to successfully create and verify the specified control graph task on their own, including users block placement, wiring, remote Dynamix pairing, graph deployment and testing of the deployed graph using the provided hardware. As shown in Figure 8, users required on average 277 seconds of training (via the tutorial) and then completed the task in approximately 122 seconds.

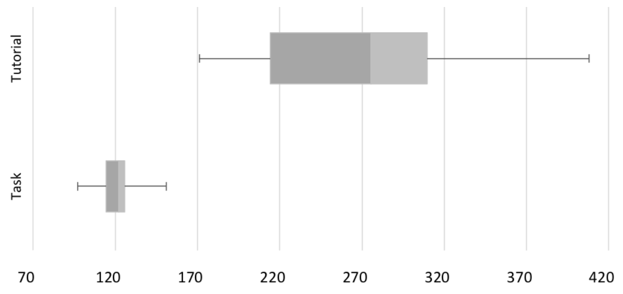

Fig. 8. Time taken by study participants to complete the tutorial and the task. The values are measured in seconds.

Many users requested the possibility to download preconfigured control graphs to use in their own configurations. However, when asked if they were interested in the ability to upload and share their own graphs to a community portal, most users were (surprisingly) not motivated to do so. This result indicates that a database of high quality, ready-made graph templates may be desirable, if provided by motivated developers and skilled enthusiasts. We are currently exploring ways of enabling better control graph publishing, discovery and integration, which will allow novice users to quickly adapt existing graphs to their particular environment through a few simple configuration steps.

After users completed the study task, they were interviewed about their experience with Ambient Flow using the System Usability Scale (SUS) questionnaire [9]. The SUS questionnaire provides a structured rating system for judging the usability of the task according to the standardized questions shown in Table I. The resulting scores represent the overall usability of the system (ranging from 0-100). A higher score represents better usability in this context. In addition, users were timed while taking the tutorial and completing the study task. Finally, users were asked several open ended questions about their impressions of the system and possible features they missed.

TABLE I. SYSTEM USABILITY SCALE (SUS) QUESTIONS [9]

Q 1	I think that I would like to use this system frequently.
Q 2	I found the system unnecessarily complex.
Q 3	I thought the system was easy to use.
Q 4	I think that I would need the support of a technical person to be able to use this system.
Q 5	I found the various functions in this system were well integrated.
Q 6	I thought there was too much inconsistency in this system.
Q 7	I would imagine that most people would learn to use this system very quickly.
Q 8	I found the system very cumbersome to use.
Q 9	I felt very confident using the system.
Q 10	I needed to learn a lot of things before I could get going with this system.

Figure 8 shows the mean answers to the SUS questions that users answered after the study task. The final results summed to a total SUS score of 81, which indicates an overall positive user experience. This was also reflected in the answers of the additional, open-ended interview questions related to their experiences and suggestions. In these answers, most users described how they liked the simplicity of the visual interface, and how they had fun drawing lines that connected functional blocks that could be then realized in the real world.

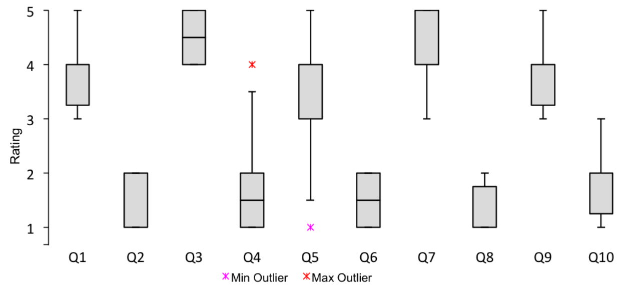

Fig. 9. Results of the Ambient Flow user study. Answer values range from 1: "Don't agree" to 5: "Strongly agree". The questions asked are displayed in Table I.

VI. RELATED WORK

There are several areas of related work that have served as inspiration and motivation for our research into orchestrating seamless interactions across heterogeneous IoT environments. In terms of promoting device interoperability, intermediate nodes, known as smart gateways [10], have been proposed as a method of encapsulating proprietary devices behind Web-based APIs. A smart gateway communicates directly with devices using appropriate service protocols and network access technologies (e.g., Bluetooth or Zigbee) and simultaneously connects to the Internet using TCP/IP, providing an in-built Web server to clients and handling data forwarding and protocol translation between Web agents and non-Web devices. As smart gateways are currently designed to solve specific network access and protocol interoperation challenges, they typically contain a static set of capabilities and cannot be extended easily at runtime [11]. Moreover, smart gateways are difficult to deploy at scale, since they must be physically installed and configured for each target network.

Related, protocol interoperation network architectures have also been developed, including ReMMoC [12]; INDISS [13]; MSDA [14]; and uMiddle [15]. Unfortunately, these approaches cannot adequately support many IoT scenarios, where wide-area deployments are critical [14]. For example, infrastructure-based approaches like INDISS and uMiddle cannot be deployed across independently administrated networks since administrative access is required. Although network overlay techniques such as MSDA do support wide-area deployment, they also require widespread adoption to be useful (i.e., each target network must have at least one participating MSDA node). ReMMoC can be used without widely deployed infrastructure; however, participating network services must be re-architected to support its Web-service discovery model. More recently, SeDiM [16] has been proposed as a client-centric interoperation solution, which provides multi-protocol translation support. While closer in spirit to our approach, SeDiM does not support the dynamic integration of new protocol support at runtime; rather, a fixed set of protocol modules can be switched on or off as needed.

Several projects have explored techniques for enabling smart space orchestration using visual programming models. Some of these systems (e.g., [17]–[19]) allow users to perform orchestration through user interfaces that expose a set of predefined rules. While such techniques are useful in scenarios that can be decomposed into an "if this then that" control structure, they are not ideal for handling streams of sensor data that directly affect actuators, such as an orientation sensor controlling the direction of a robotic device. In these cases, our flow based approach is more flexible, as it covers "if this then that" scenarios as well as stream-based actuator control in a unified manner.

Finally, other visual approaches assume that every smart device will (and can) be represented by a unified specification schema, like the Resource Description Framework (RDF) in the case of [20] and [21]. We acknowledge that it is generally a good idea to use existing standards to describe smart device services, but we are hesitant about the choice of RDF, which tends to be become bloated and cumbersome when used to describe real-time data such as sensor streams. Semantic approaches will likely play an important role in large-scale orchestration scenarios, such as city-wide service coordination. In contrast, our approach focuses on orchestrating spontaneous interactions with potentially unknown smart devices situated in the user's environment, which may be encountered at runtime. As such, our approach can be understood as complementary to current orchestration techniques, as well as others.

VII. CONCLUSIONS AND OUTLOOK

In this work, we presented a novel set of smart space design tools that enable non-programmers to visually "remix" ambient environments in new, playful and potentially unforeseen ways. The approach, called Ambient Flow, builds on our previously presented Ambient Dynamix and Ambient Control technologies, which transform a user's commodity mobile device into adaptive smart gateway that provide service adaptation and protocol translation services through plug-ins that can be installed on-the-fly. During design time, Ambient Flow tooling can be accessed from a desktop computer (or tablet) through a conventional Web browser. Live information from a remotely paired Dynamix instance is used to render discovered connected devices as block diagrams that can be "wired" together using an intuitive flow graph model. Completed control graphs can be sent from the browser back over the network to the Dynamix-based device, where they are setup by Ambient Control in real time for testing and debugging. Finalized smart space designs can be published to a public online repository (or private local server) together with contextual scoping tags (e.g., radio beacon or geo-fence data). During runtime, mobile users to discover and use shared designs when situated within the specified context using only a Dynamix-based device.

We envisage this type of tooling as useful for new types of design professionals that will likely emerge as the IoT continues to expand into everyday environments. Such "ambient designers" will likely prefer to focus on creating innovative user experiences rather than solve low-level IoT interoperability issues. Accordingly, Ambient Flow lowers the complexity of smart space orchestration through efficient user guidance and automated configuration.

In terms of evaluation, we produced a fully operational prototype system and conducted a preliminary user study. In the study, 10 participants utilized the Ambient Flow prototype to visually connect a Sphero robot to a Hue network light in such a away that the orientation of the robot controlled the color of the Hue light. Users were also asked to deploy the completed configuration to a remotely paired Dynamix instance for realization. We found that all users were able to successfully complete the study task of creating a new control graph on their own, which in itself is a very encouraging result. The final System Usability Scale result for all participants was 81, which indicates an overall positive user experience.

We are focusing on several areas of future work. We are continuing to wrap smart device functionality within open-source Dynamix plug-ins that adhere to the Ambient Control methodology. We feel that a large pool of plug-ins is necessary for exploring the real-world implications of our architecture. We are also developing more robust remote pairing capabilities for Dynamix, including support for encrypted remote communications. Finally, we are investigating how control graphs can be published publicly and loaded automatically based on contextual triggering in large-scale IoT environments.

References

[1] D. Carlson, B. Altakrouri, and A. Schrader, "An ad-hoc smart gateway platform for the web of things." in *IEEE International Conference on Internet of Things (iThings 2013)*. IEEE Computer Society, Conference Proceedings.

[2] D. Carlson and A. Schrader, "Dynamix: An open plug-and-play context framework for android," in *Internet of Things (IOT), 2012 3rd International Conference on the*, Oct 2012, pp. 151–158.

[3] M. Pagel and D. Carlson, "Ambient control: A mobile framework for dynamically remixing the internet of things," in *IEEE International Symposium on a World of Wireless Mobile and Multimedia Networks*. IEEE, 2015, Conference Proceedings.

[4] D. Carlson and M. Pagel, "Tap to interact: Towards dynamically remixing the internet of things." in *Eleventh Annual International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous 2014)*, 2014, Conference Proceedings.

[5] D. Carlson, B. Altakrouri, and A. Schrader, "Ambientweb: Bridging the web's cyber-physical gap," in *Internet of Things (IOT), 2012 3rd International Conference on the*, Oct 2012, pp. 1–8.

[6] J. Morrison, *Flow-based Programming: A New Approach to Application Development*. J.P. Morrison Enterprises, 2010.

[7] F. Oliphant *et al.*, "Meemoo: Hackable web app framework," 2012.

[8] D. Carlson, B. Altakrouri, and A. Schrader, "Reinventing the share button for physical spaces," in *IEEE International Conference on Pervasive Computing and Communication (PerCom 2013)*. IEEE, 2013, Conference Proceedings.

[9] J. Brooke, "Sus: A quick and dirty usability scale," 1996.

[10] D. Guinard, V. Trifa, and E. Wilde, "A resource oriented architecture for the web of things," in *International Conference on the Internet of Things (IoT 2010)*. IEEE Computer Society, Conference Proceedings, pp. 1 – 8.

[11] D. Zeng, S. Guo, and Z. Cheng, "The web of things: A survey," *Journal of Communications*, vol. 6, pp. 424–438, 2011.

[12] P. Grace, G. S. Blair, and S. Samuel, "Remmoc: A reflective middleware to support mobile client interoperability," in *The 5th International Symposium on Distributed Objects and Applications (DOA 2003)*, Conference Proceedings, pp. 1170–1187.

[13] Y.-D. Bromberg and V. Issarny, "Indiss: Interoperable discovery system for networked services," in *ACM/IFIP/USENIX 6th International Middleware Conference*, vol. 3790. Springer Berlin Heidelberg, Conference Proceedings, pp. 164–183.

[14] P.-G. Raverdy, O. Riva, A. d. L. Chapelle, R. Chibout, and V. Issarny, "Efficient context-aware service discovery in multi-protocol pervasive environments," in *Proceedings of the 7th International Conference on Mobile Data Management*. IEEE Computer Society, Conference Proceedings, 11366913.

[15] J. Nakazawa, W. K. Edwards, H. Tokuda, and U. Ramachandran, "A bridging framework for universal interoperability in pervasive systems," in *The 26th IEEE International Conference on Distributed Computing Systems*. IEEE Computer Society, Conference Proceedings.

[16] F. Carlos, G. Paul, and S. B. Gordon, "Sedim: A middleware framework for interoperable service discovery in heterogeneous networks," *ACM Trans. Auton. Adapt. Syst.*, vol. 6, no. 1, pp. 1–8, 2011, 1921647.

[17] M. García-Herranz, P. A. Haya, and X. Alamán, "Towards a ubiquitous end-user programming system for smart spaces." *J. UCS*, vol. 16, no. 12, pp. 1633–1649, 2010.

[18] Z. Drey and C. Consel, "A visual, open-ended approach to prototyping ubiquitous computing applications," in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference on*, March 2010, pp. 817–819.

[19] F. Perez, P. Valderas, and J. Fons, "Allowing end-users to participate within model-driven development approaches," in *Visual Languages and Human-Centric Computing (VL/HCC), 2011 IEEE Symposium on*, Sept 2011, pp. 187–190.

[20] N. D. Rodríguez, J. Lilius, M. P. Cuéllar, and M. D. Calvo-Flores, "Extending semantic web tools for improving smart spaces interoperability and usability," in *Distributed Computing and Artificial Intelligence*. Springer, 2013, pp. 45–52.

[21] M. Vega-Barbas, D. Casado-Mansilla, M. A. Valero, D. López-de Ipina, J. Bravo, and F. Flórez, "Smart spaces and smart objects interoperability architecture (s3oia)," in *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on*. IEEE, 2012, pp. 725–730.