

ZUMA: A Platform for Smart-Home Environments*

C.R. Baker, Y. Markovsky, J. van Greunen, J. Rabaey, J. Wawrzynek, A. Wolisz[#]

University of California, Berkeley, USA
{crbaker, yurym, janavg, jan, johnw, wolisz}@eecs.berkeley.edu

Abstract: Although electronic devices permeate the home and offer unsurpassed power and features, many impediments still exist to realizing the concept of the smart-home: configuration complexity, minimal device interoperability, difficulty of use, lack of personalization, and no integration with sensors for awareness and adaptation. We define four tenets of smart-home environment: Zero-configuration, Universality, Multi-user optimality, and Adaptability (ZUMA), and describe a platform based on a set of clean abstractions for users, content, and devices. The platform enables configuration and organization of content and networked heterogeneous devices in a smart-home environment. We validate the platform with a prototype implementation and analyze its flexibility and infrastructure requirements.

1 INTRODUCTION

Falling electronics prices have brought sophisticated consumer electronics within the reach of average users. Further, due to the proliferation of different device types, communication, and media coding standards, interoperability between devices will become increasingly important. Advances in the electronics industry have also driven growth in several related areas, in addition to home entertainment, including surveillance, home automation products (e.g. lighting, heating, and power), home appliances (e.g. laundry machines, fridges, etc.), and ad-hoc wireless sensor networks (AWSN), which promise to add a truly ambient intelligent component to the home. The future home clearly represents an opportunity for the convergence of these different technologies far beyond the level of integration seen today. In fact, the future smart-home will contain a collection of heterogeneous networked devices, capable of distributed computation, dynamic reconfiguration, high-performance media dissemination, and user/environment awareness. There are several problems that need to be addressed for this convergence to be successful.

In the past, electronic equipment was sold as vertically integrated boxes. Today, new solutions and standards are introduced faster than the lifetime of many pieces of infrastructure. As a result, consumers incrementally upgrade their existing systems. In addition, new products

are introduced in the sensor network, security, and home automation spaces, and must be added to the existing system. Thus, it is desirable to design a system that can seamlessly integrate new devices *without* user intervention. The solution outlined in this paper supports automatic integration of new devices and efficient use of the functionality they provide. We call this *Zero-configuration*. (Here used more generally than the use by IETF Zeroconf Working Group in their effort toward automatic simple-network IP configuration).

As alluded to earlier, the interoperability of devices is complicated by the disparity at several levels of the networking stack: different wireless technologies (e.g. sensor motes vs. 802.11), device communication protocols, and presentation standards (e.g. video encoding). The agreement of a common set of standards is unlikely; therefore, methods must exist that allows the connection of any set of devices to any other set of devices. The role of a smart-home environment is to provide the necessary levels of indirection to realize this goal. We call this *Universality* and our solution addresses this problem.

Another issue occurs when different users share devices in the home. These users often have different preferences and modes of usage. Computers have dealt with the personalization issue by allowing the creation of separate accounts. However, it is not currently possible to personalize settings for an entire space or home entertainment application (e.g. light levels, speaker volume, etc.) and resolve conflicts between users. The system should recognize and accommodate different users according to their properties, permissions, and preferences.

Furthermore, the explosion of different user interfaces presents a nightmare to those wishing to use the current devices in the home. Consider a modern home that has a multi-media center with a television set, DVR, DVD player, a sound system, lighting control and a security system. In the worst case, the user requires a remote control for each device. It is inconvenient and often confusing. The system should enable applications to provide a uniform manner of control for simple manipulation of the home environment. The presented solution will address the issues of enabling personalization and easy manipulation of the environment. We refer to this as *Multi-user optimality*.

Current home environments are relatively static, meaning their operation requires manual intervention. It has long been a goal of researchers to make the devices and applications dynamically adapt to changes in the environment.

*The ZUMA project is funded by the Gigascale Systems Research Center www.gigascale.org

[#]A. Wolisz is affiliated both with TU Berlin and UC Berkeley

One would like adaptation to the presence of humans, their minute-to-minute desires (*i.e.* both implicit or explicit), and the current environment conditions (*e.g.* a new device arrives). The solution this paper discusses will enable such dynamic run-time reconfiguration that we call *Adaptability* of the system.

Based on these four tenets, we propose ZUMA (Zero-configuration, Universality, Multi-user optimality, and Adaptability), a platform for smart-home environments. Important issues such as digital rights management, user privacy, and network security are beyond the scope of this paper. We do not preclude the existence of these technologies in our system, however, solutions are either under development by various industry/research groups or are not appropriate for academic research.

1.1 Scenario

We present a scenario that illustrates a subset of the desired behavior of ZUMA platform. A multi-media application was selected for this scenario because with currently deployed technology, it allows the most concrete use case. In this work, the following scenario will serve as a vehicle for discussion and for differentiating ZUMA from other proposed solutions:

John walks into a room and turns on the television, which starts a multi-media viewing application. John likes to watch violent sports, and according to his preferences, the TV automatically switches to WWE wrestling. The WWE wrestling channel comes in from the set-top box. John watches for awhile, but then his son Adam walks in with a laptop. Adam is a minor, and is not allowed to watch WWE. The television immediately stops showing WWE and puts up a menu to select different content.

In the background, the system continues recording the WWE channel and stores the time at which John stopped watching. This allows John to resume watching WWE at a later time. John now selects "Finding Nemo" (from Adam's laptop) to watch with Adam. "Finding Nemo" starts playing on the television.

This scenario demonstrates aforementioned tenets. *Universality*: the television, a set-top box, and a laptop inter-operating effortlessly together. *Adaptability*: the system detects non-electronic events (such as the movement of people). This "ambient" functionality is provided by sensor networks. Additionally, content is decoupled from the rendering devices. In order to render content on different devices, it may need to be reformatted and/or trans-coded. *Zero-configuration*: as new devices (*e.g.* the laptop) enter an environment, they are seamlessly integrated. *Multi-user optimality*: The system abides by the preferences and permissions of a scenario's users.

1.2 Contributions and Outline

The main contributions of this paper include:

- a set of abstractions, specifically for those elements affecting people, content, and system capabilities, which drive a platform (Section 2),
- a platform to enable configuration and organization of content and networked heterogeneous devices in a smart-home environment (Section 3),
- a prototype implementation to demonstrate the platform and infrastructure requirements (Section 4).

We will discuss related work in Section 5 and conclude in Section 6.

2 ZUMA ABSTRACTIONS

As described in Section 0, the convergence of different applications, devices, and networks in the home presents interesting research challenges. This section addresses one of those challenges in defining a set of abstractions to drive the design of a ZUMA. In this section, we assume the existence of a "user" and an "application," which are described in detail in Section 3. The user is defined as an active entity that can instantiate applications—programs describing purposeful functionality.

The *environment* is the umbrella concept in the platform. Intuitively, the environment contains consumer electronics (DVD player, computer, hand held tablet PC, cellphone, television, stereo), sensors/actuators (motion, light, temperature), appliances (heating/cooling system, lighting, cameras), subscription services/devices (television, Internet, security, power, telephony), infrastructure (disk storage, routers, bridges, wireless access points, compute nodes), and people who may make use of the elements (you, me, guests). The environment contains virtually every piece of data and device at the user's disposal. Together, all of this is too overwhelming for any person or algorithm to efficiently utilize.

A subset of this environment that interacts with an application in a specified scope (defined below) is known as the *current environment*. The current environment can be logically considered a set of services. The questions such services can answer may include: "what is user X's position?" "what devices exist?" "what movies are available?" "which device near my current position renders video?" "what is the temperature?" *etc.* These services can also notify of events such as "user X has changed location," "a new device is available," "a movie is no longer available," "the temperature has reached X," *etc.* In other words, the current environment offers query and subscription services.

A *scope* bounds the environment and it can be spatial, temporal, or based on meta-data. Example scopes include simple and familiar concepts such as "my house", "my

room”, but may also include “two rooms on the opposite sides of a planet” with participants of a teleconference. In the presented scenario, the scope is the current room where John and Adam are located. The current environment can be characterized in terms of *personae*, *capabilities*, and *content*. These elements are discussed in the following paragraphs with appropriate references to our scenario.

2.1 Personae

The importance of the persona is to capture user-specific details that affect the user’s interaction with content and capabilities, thus enabling multi-user optimality. A persona may represent a single person, groups of people, or organizations (e.g. fire department). A persona contains preferences, permissions and properties. Preferences represent personal choices captured in some meaningful way to the system. In addition, a persona with sufficient privilege may set rules of operation for the environment (e.g. parent limits speaker volume to X in home). In essence their virtual presence is enough to enforce their preferences. Permissions are user’s access rights to devices and content, based on content attributes and metadata. For example, permissions may limit user’s ability to watch rated-R movies or control security related devices. Finally, a persona’s properties contain information that describe user(s), e.g. birth date, passwords, and other data. The run-time system interprets, acts on, and resolves conflicts between multiple users with, personae.

2.2 Capabilities

Capabilities uniformly capture physical resources (devices, network, programs), in the environment. The resources are exposed as a capability by the functionality they provide (e.g. radio – *audio creator*, TV – *video renderer*, temperature sensor – “*current temperature*” *content source*, router – *connection from A to B*, transcoder – *conversion from format X to Y*). Capabilities may abstract familiar devices such as: set-top box, DVD player, remote control, computer, stereo system, security system, lighting and appliances, energy monitors, as well as the unfamiliar: sensor arrays, hidden compute devices, smart materials and contextual control objects.

The scope that defines the current environment, sets spatial and temporal conditions for availability of certain capabilities. An interface language describes each capability for all devices in the home. Integrating a new device simply reduces to exporting a set of capabilities using this language, which the system recognizes and uses appropriately. New capabilities may be may be 'understood' by the system through an online database. Through capability (and device) discovery, the run-time system enables simple integration (i.e. the zero-configuration in ZUMA) of new devices.

2.3 Content

While capabilities abstract the physical world, content abstracts information that capabilities can manipulate. Content may represent a range of data: media streams, sensor readings (light, temperature, motion, identification), security information, energy monitoring results, etc. The ZUMA platform treats all content uniformly through typing of content and capabilities, which allows their straightforward manipulation and trivial integration of new contents. For example, classifying a video as MPEG4 and a TV input as MPEG4 simplifies their association when a user request is implemented. Integration of a new piece of content only requires a matching capability (a compatible interface description), further enabling zero-configuration. Both content and capabilities find a common ground under this description language (of which full treatment is beyond the scope of this paper and will be subject of future research).

In some cases, content and capabilities are tied to a common physical resource. For instance, a capability to play a video and the content from a DVD track, are provided by the DVD player. Similarly, a temperature sensor provides a capability to emit temperature and the “temperature” content. However, this tie between content and capability does not always exist. For example, devices that transform (e.g. picture-in-picture) or render (e.g. speakers and video screen) may operate on a range of content. Furthermore, many capabilities may be aggregated to operate on or produce a single piece of content (e.g. all temperature sensors in a room provide an average room temperature).

3 ZUMA PLATFORM

This section describes a platform that enables configuration and organization of content and networked heterogeneous devices in a smart-home environment. Figure 1 shows the layers of the proposed platform. The users interact with sessions; sessions deliver behavioral-level tasks to the ambient OS. Ambient OS continuously maps these tasks into implementation-level tasks (physical resources). The session and mapping processes are closely aware of the changes and events in the current environment and adapt their operation accordingly.

The communication between the sessions and Run-Time Continuous Mapper (RTCM), and the RTCM and the Distributed Run-Time System, is called out explicitly to illustrate the mapping process between behavioral-level and implementation-level tasks. Communication between the current environment service and aforementioned components is implied by adjacency of layers in the diagram.

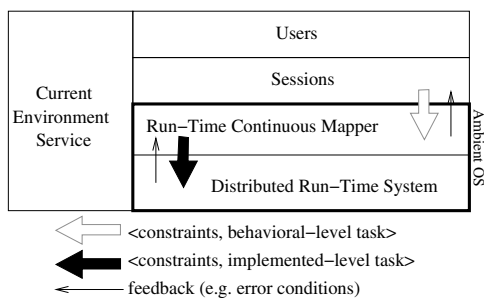


Figure 1: The platform comprises users, sessions, run-time continuous mapper-and-system. The Current Environment Service is used by all layers.

3.1 Environment Services

The environment is dynamic and is exposed as a set of query and subscription services (Figure 2), which other active entities use. Entities query the environment within a scope. The service uses scoping to create a *current environment*. There are two types of service classes: (1) higher-level services (e.g. available content, capabilities, and personae), and (2) lower-level information (e.g. available devices and locations). Typically, sessions will use the high-level service to query capabilities (e.g. video rendering), content (e.g. set-top box content, DVD tracks), and personae (e.g. John and Adam). On the other hand, the run-time system needs low-level information.

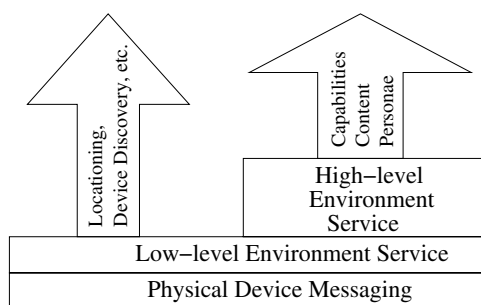


Figure 2: The details of the current environment services.

3.2 Users

The user, which corresponds to a persona, is an active entity that initiates and manipulates (interact, suspend, resume, or shutdown) sessions. A user can be associated with or own many sessions. The session owner has full permission to interact with the session and can explicitly or implicitly set/limit permissions of others to interact with the session. For example, the head of a household owns the security monitoring session, and will grant interactive access only to the police. Users may initiate and manipulate sessions through application UI, depending on the way the application has defined such functionality (e.g. through a local display using a keyboard and mouse, a remote control, or through a contextual object).

3.3 Session

A *session* is an instantiation (active entity) of an *application*, akin to the analogy between a process and a program in an operating system. The session executes the application's functionality on the behalf of a user. The session is responsible for managing UI and monitoring environment dynamics through query and subscription mechanisms of the environment services. A session is the key to run-time adaptability and the ease of use demanded by multi-user optimality.

To realize user intention, the session generates behavioral tasks with appropriate constraints, which specify desired content manipulation. Figure 1 shows session instantiation (1) and the task generation (3) from user interaction. In the scenario in Section 1.1, the multimedia-watching session generates, based on John's preferences, an initial task that specifies the rendering of WWE wrestling on a local TV. The top of Figure 2 shows the initial behavioral task for the scenario example. The following sections will discuss the session state generation (2), and will elaborate on tasks.

The session scopes the environment at run-time, which yields the current environment. The current environment contains only information relevant to the session. Each session has a unique view of the environment. For example, in our scenario, the scoping only includes the living room and all people who are in the room.

Many sessions may operate concurrently and contend for the same resources. If there is a conflict between sessions, resolution is performed by the Ambient OS (Section 3.4). Conflicts in resource allocation do not always result in one session successfully attaining the resource and the other not. The Ambient OS can intelligently map multiple sessions onto the same resource(s) and combine their content. For example, a home-theater session is using the living room TV, and the home security session must display the live feed from the front door camera on the same TV. The Ambient OS can resolve the conflict by rendering the two streams as picture-in-picture. This also simplifies the application because it does not need to negotiate for resources with other sessions.

The following example sessions may run concurrently:

- **Home Security:** devices include outdoor cameras/lighting, window/door sensors; users or personae include fire, police, and homeowner.
- **Home-entertainment:** devices include display, audio, and lighting; users include people in the house. Simplified, limited TV-watching sessions may also exist in the kitchen or bedroom.
- **Home-automation:** devices include the home appliances (heating, lighting, air purification). Users include authorized family members.
- **Health monitoring:** devices include wearable sensors, capabilities—show/monitor vitals; us-

ers/personae include emergency medical response, health-care provider, and home residents.

Session State Users can suspend and resume sessions. To support this, we need a notion of recent history and therefore, as the session executes, it generates state (Figure 1, step 2). The state by itself is a passive entity (no different from stored content), just as a current web browser's history is passive until the user returns to it with BACK button. Intuitively the session state is a log of events and checkpoints. The session (or the run-time system) continuously logs its current state and creates periodic checkpoints to limit the log size and permit fast recovery. The state representation can be arbitrarily complex depending on a supported feature set. The state may include history and current location in a piece of content being rendered, a description of user intention (behavioral tasks), actor state (e.g. Sequence operator position), and the set of constraints. To resume a suspended session, the log is retrieved, and the latest checkpoint is used to restore state.

The notion of session state allows us to perform two powerful operations on sessions: *relocation* and *duplication*, both with subsequent remapping to the environment.

Relocation refers to the process during which session state is transferred to a different environment, following a user. For example, if a user is watching a movie in her home and then leaves the home to ride in a train, the session state containing the user's position in the movie, and other attributes, can be automatically transferred along with the user to the new environment. If the movie is available in the new environment (i.e. possibly it was automatically transferred to her laptop), then the user can resume the movie at the same point with the same preferences and setting.

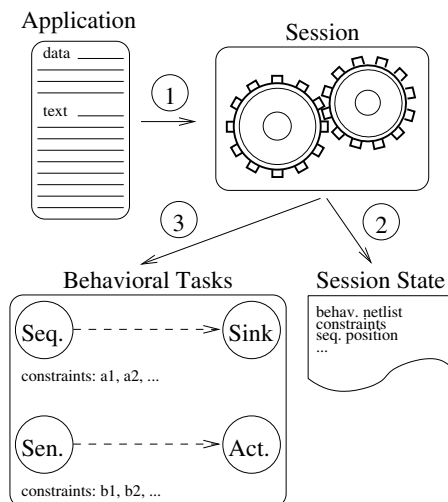


Figure 1: An application instantiation (1) produces a session, which produces state (2) and tasks (3). The task contains a Sequence, sink, Sensor, and Actuator actors.

Duplication of a session may occur when personae that shared a session diverge. Consider the following duplication scenario. Two people are watching a movie in a home-theater session. One person leaves the room and finishes watching the movie in his bedroom. The properties both users desire, is to continue watching from the same instant in the movie, and to control their sessions independently (i.e. a pause command in the living room should not pause the movie in the bedroom). By duplicating the session state, each user gets their own session.

Behavioral Task A session generates behavioral tasks that capture the high-level, user-specified functionality and intent, but do not contain the details required for an implementation.

Figure 1 shows the session generating two behavioral tasks: (1) a sequence actor (an ordered list of content with current position) streaming content to a sink (e.g. video stream of home theater application); (2) a sensor actor streaming content to an actuator (e.g. lighting control). The behavioral task, along with relevant constraints (e.g. user preferences, permissions, QoS) is passed from the session to the Ambient OS (Figure 1). Feedback from the Ambient OS is in the form of error conditions, such as violation of constraints (e.g. failure to meet QoS) or failure to fulfill user intention (e.g. resource unavailable).

A session produces behavioral tasks that reflect user intent. These tasks are dynamic in nature, constantly updated to reflect changes in the current environment. As mentioned previously, the session is responsible for querying and subscribing to environment services. By using these services, the session is aware of the events relevant to its operation and can adapt the behavioral tasks to reflect these changes. Furthermore, it can request a remapping of the task by the Ambient OS.

The behavioral task contains both the *control logic* and the *content flow* between capabilities: sources and sinks, filters (e.g. convert to black and white to simulate vintage video) and transforms (e.g. picture-in-picture). We represent behavioral tasks as flow-graphs, with capabilities as graph actors, and directed edges as content flow and control. In the Section 1.1 scenario, the behavioral task contains one content element (e.g. WWE wrestling program) and a sink actor (e.g. local TV), corresponding to the video source and renderer, respectively (Figure 2).

3.4 Ambient OS

The session creates behavioral-level tasks and sends them with a set of constraints to the Ambient OS, which consists of the real-time continuous mapper (RTCM) and a distributed run-time system (DRTS)

Real-Time Continuous Mapper The real-time continuous mapper (RTCM) transforms the behavioral task into an implementation task abiding by the constraints specified by the session. The mapper enables platform adaptability and universality by mapping behavioral tasks to implementation tasks with the required elements for de-

vice interoperability (e.g. trans-coding). [14] discusses universality via the Universal Content Router (UCR).

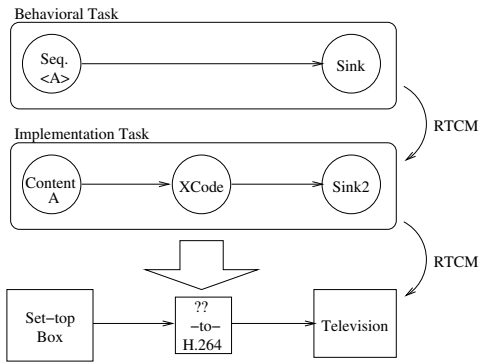


Figure 2: Initial mapping of scenario's behavioral task to implementation task.

The RTCM is a logically continuous process: it constantly re-maps the required tasks to reflect any changes in the tasks or the current environment. The match between the requested elements of the task and the devices must be exact (*i.e.* legal), otherwise the mapping process will fail. Of course, legal mappings are dependent on the constraints of the session and the persona's preferences: a persona may accept degraded video quality when transferring a video session from the HD-TV to a handheld TV or a task may be amenable to time-multiplexing (e.g. of a compute node for non-real-time elements of a task) if resources are scarce. Through a subscription service, the RTCM provides feedback to the application to indicate success, alert of changes to the environment, and indicate an error or failure to meet a constraint.

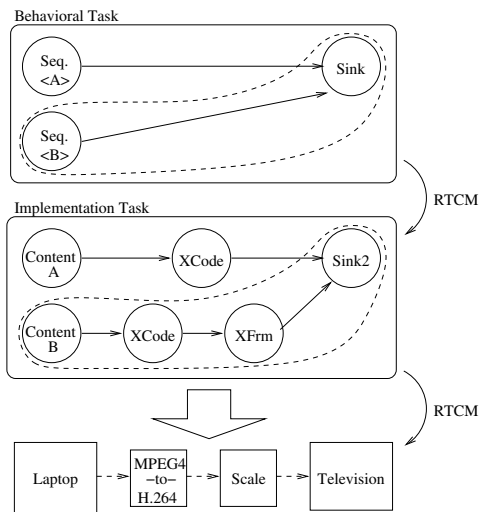


Figure 3: Remapping of scenario's behavioral task into an implementation task and then onto physical resources.

Figure 2 illustrates the mapping process (from top to bottom of the figure) of a behavioral task to an implementation task and the resource assignment to the implementation task. The mapping process begins with behavioral task elaboration. RTCM performs a type-check of content and capability actors to ensure that output and input types along each edge match. In Figure 2,

the type of the sequence actor does not match the type of the sink. Therefore, the mapper inserts a trans-coding actor (Xcode). Next, the RTCM attempts to map the elaborated task assuming that all accessible devices are available. Here the goal is to meet all relevant mapping constraints, regardless of resource availability, to verify that the elaborated task is implementable. If this is not the case with these loose requirements, immediate feedback is provided to the session.

The RTCM continues the mapping process by resolving and negotiating resource use between all sessions and their respective tasks. Here persona preferences and permissions are combined in attempt to resolve allocation conflicts and incompatibilities between communicating flow-graph components (e.g. bit-rate, QoS, connectivity). This process creates implementation tasks (middle of Figure 2), which together meet *all* constraints and optimize objectives specified explicitly by users and implicitly by all personae in the environment. Finally, the RTCM maps the implementation tasks to physical resources (bottom of Figure 2) and produces a detailed allocation and configuration map. For most end devices this mapping is relatively simple (and each runs a subset of the task's functionality according to their exported capabilities), however, allocation of CPU cycles, reconfigurable hardware, and configurable radios, *etc.* will be implemented on the universal content router(s) (see [14] for additional details).

To illustrate how the mapping process works under the dynamics of moment-to-moment user manipulation and changes in the current environment, consider our scenario from Section 1.1. Suppose that the presence of Adam suspended the current multimedia-watching session. After John selects a new piece of content to view (movie "Finding Nemo" from Adam's laptop), the behavioral and implementation tasks undergo the changes illustrated in Figure 3. The new content requires different trans-coding, as well as a transform to fit the sink. After these changes to the implementation task, the resources are re-allocated, and the session resumes execution.

Although the discussion of RTCM may imply a centralized mapping process, a *distributed* implementation would be a preferable, scalable approach. Geographically and temporally delineated environments, people's preferences, permissions, and priorities subset resources, content and personae that are involved in a given mapping decision. These non-intersecting subsets enable us to parallelize and distribute RTCM operation across a range of compute devices. A scenario where a single mapping decision involves all or nearly all content, devices and personae will be rare in practice. Section 4 contains details of our prototype mapper.

Distributed Run-Time System To execute a mapped implementation task, the system must perform a range of device-specific resource management operations: hardware reconfiguration, process load-balancing, reservation of connectivity to provide required quality-of-service and bit-rates, *etc.*

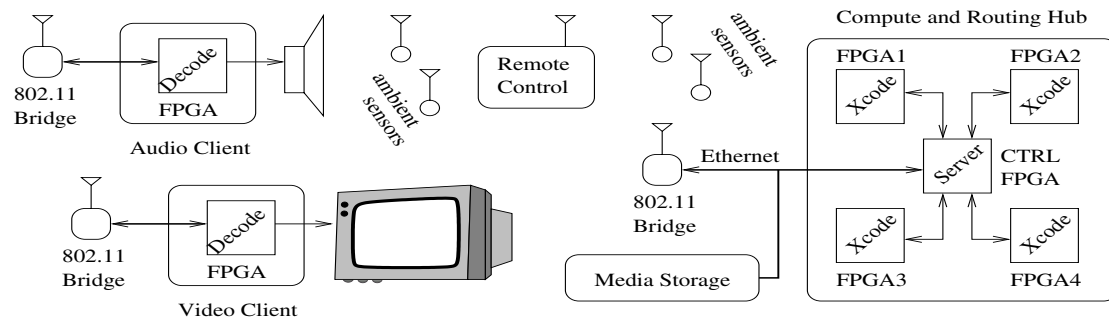


Figure 4: The prototype contains the compute and routing hub, audio/video, storage devices and the remote control.

The Distributed Run-Time System (DRTS) handles these critical, but routine and non-portable tasks. The DRTS is a software layer running on all devices, which provides the necessary level of indirection between the middleware layers above and the bare devices below, thus enabling the ZUMA property of universality. Localizing device dependent functionality in DRTS makes the platform easily extensible to new devices, and mechanisms for device capability discovery enables zero-configuration.

DRTS imposes a minimal requirement that all devices to support a common device control protocol. Those devices that cannot support a common protocol can be bridged (e.g. temperature sensor can be bridged through a “smart” base station). The control protocol must enable device discovery and basic management only. Advanced control protocol features only need to be supported by those devices that need them. From this basic protocol, the DRTS controls the devices and executes the implementation tasks. The DRTS also exports information to the environment services for querying by higher-level layers in the platform stack.

4 PROTOTYPE

4.1 Overview

We have developed a prototype to validate the proposed platform. Although a full implementation is beyond a scope of an academic project, the prototype includes all basic features and comprises a compute and routing hub, video and audio clients, media storage devices, remote control and sensor devices (Figure 4). These devices share a common IP network, supported by the wired and wireless connections.

The content sources include storage devices that stream audio and video content upon request. The sinks are video and audio clients that decode and render incoming streams. The decoding is performed either in software or in Field Programmable Gate Arrays (FPGA) hardware depending on the real-time computational requirements. FPGAs are post-fabrication programmable integrated circuits that contain microprocessor cores (ideal for low performance, complex control) and configurable logic

(ideal for high performance, minimal control computations).

Further, the prototype has compute nodes that can be dynamically programmed by the run-time system to execute particular operations. For example, if the source content format does not match the sink format (e.g. Motion JPEG video source and MPEG4 video renderer), the run-time system allocates an FPGA and programs it to trans-code Motion JPEG into MPEG4 in real time as the video is streamed from its source to destination.

4.2 Hub Server

The prototype is centrally managed by the Hub Server, the run-time system that manipulates devices, session state, content listing, and handles discovery. In addition to these routine tasks, the run-time system contains a continuous mapper, which “re-implements” the sessions as requested by user. The mapper automatically deals with several types of constraints: mismatch in content type between the media source and sink, and change of user or device location. Whenever user makes a request through a remote control or changes his/her location, the run-time mapper re-evaluates specified tasks to meet all constraints (Figure 3).

The structure of the Hub Server mimics that of the previously discussed Ambient OS (Figure 5). Our focus was to the make Hub Server implementation simple and manageable, to allow us to focus on control algorithm development, rather than performance optimization. The Server consists of two types of threads: client service threads and a mapper thread. A client service thread is responsible for a single client, (i.e. a content source, sink, remote control, etc.). The service thread receives queries and requests from its client and converts them into mapper events, which are added to the queue.

The mapper thread de-queues events and handles them by the pipeline of Behavioral Task Mapper, Implementation Task Mapper, and Device Control Layer, which implement a large subset of functionality of RTCM and DRTS described in Section 3.4. Before we examine the layers, let us summarize the key data structures.

- **Task Table** contains active behavioral tasks, including owner and session ID, task description, task state (e.g. the current play position).

- **Device Table** contains active devices, including I/O signature, location, type and capabilities.
- **Location Table** maps users/devices to locations.
- **Content Table** maps content to a source device.

The remote control runs the session, which sends task creation and maintenance requests to the mapper. The Behavioral Task Mapper (BTM) creates, initializes, and modifies behavioral task state, runs the tasks and shuts them down upon user's request or errors in the system. The BTM handles “high-level” requests originating from the users and/or sensors, such as PLAY, STOP, REW and other requests as well as changes in user locations. The BTM produces implementation tasks, which are simple control-less tasks containing a specific piece of content currently being rendered, user-requested transformations and rendering sinks.

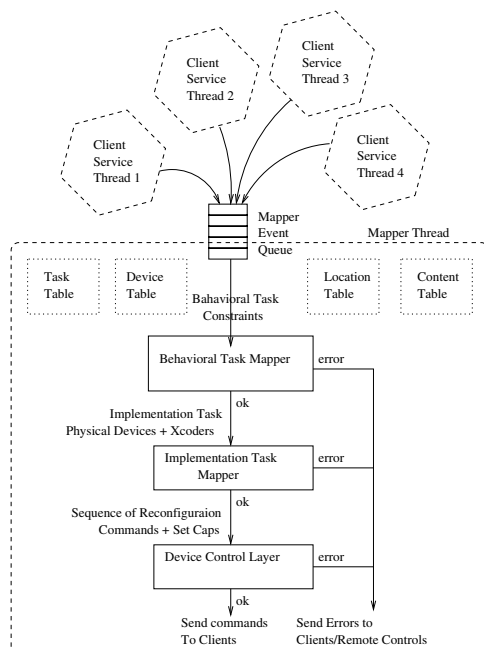


Figure 5: Hub Server contains two types of threads: client service thread and mapper thread.

The Implementation Task Mapper (ITM) converts implementation tasks into device control commands. In practice, since the sessions are continuously running, the implementation tasks produced by the BTM for the same behavioral task are not completely distinct from one another, but share common actors. For example, as the behavioral task goes from one piece of content to the next, the only component that could change in the implementation graph is the source node and/or its associate content name and possibly the trans-coders. Therefore, the Implementation Task Mapper (ITM) supports *incremental* operation. Given an implementation task, the ITM identifies the differences between the old task and the new one, and produces a sequence of commands to convert the old implementation graph into the new one. To produce the device control commands, the ITM (1) elaborates the task by inserting a content trans-coder if there is a mismatch between the output and the input in a

connection; and (2) allocates resources for each of the nodes in the implementation task. If both of these steps succeed, generation of device control commands is straightforward. The lower Device Control Layer handles device specific control and management. Although, ideally all devices should be treated uniformly, it may not always be possible. The device control layer abstracts the device differences and greatly simplifies implementation of the upper mapper layers.

The mapper creates an illusion of “continuous” remapping of sessions and tasks, but in practice, the system remaps only when the following events occur:

- **Create Behv Task** – the mapper adds the task to the Task Table and maps it. Session owner is notified if errors encountered.
- **Delete Behv Task** – the mapper terminates the task and shuts down devices allocated to it and notifies the session owner.
- **Register Device** – a device came on-line. Session and task remapping may be required, in case the user requested to render a piece of content on the *closest* to him device, and the new device is the closest.
- **Un-Register Device** – a device disconnected. The affected sessions are remapped; those that *require* this device are shutdown, and the owners are notified.
- **User Location Change** – remap all sessions affected by the user (via ownership or constraints).
- **Content Unavailable** – a content source disconnected. Affected sessions are remapped; session owners are notified.
- **Device Notify** – a client sent the Ambient OS a notification: “End of File” or “Location Change”. This event is further discussed below.
- **Set Session Caps** – a remote control request to modify the session (*e.g.* play, stop, *etc.*). The mapper may remap the session and then forward the request to the device.

Let us elaborate on Device Notify event, which is central to the prototype. A content source sends the “End of File” notification to Hub Server, when the source reaches the end of the file being streamed. The event triggers a sequence of actions in the mapper: (1) the BMT switches the Sequence Source Actor to the next piece of content in the sequence, and emits the corresponding implementation task; (2) the ITM remaps the task, and (3) control commands are sent to the content source and any nodes responsible for content trans-coding to “realize” the task.

Handling the “Location Change” notification is similar. User location change is detected by one of our “human presence” pressure mats and reported to the Hub Server. The mapper remaps the behavioral tasks with the rendering sink constrained to the position of the user. The remapping process is similar to (1) – (3) above: the sink

device may be updated as a result of the user location change, which may result in a change in the trans-coder.

Device Control Protocol To control a range of heterogeneous devices, we have developed a light-weight Device Control Protocol (DCP), which contains several simple commands that every device is required to implement to be a part of the system. These commands enable discovery of devices, capabilities, their locations and available content. The remainder of the DCP commands can be safely ignored by all, but relevant devices: creation/destruction/modification of sessions are used primarily by the remote control; capabilities query and set are used only by a device offering a capability.

The discovery mechanism implemented in the prototype is simple. Every device on start-up registers with the Hub Server. Each device declares its capabilities, location and any content it provides. A remote control device can query the run-time environment to determine what content and content renderers are available at a particular. A remote control can also create behavioral tasks, which typically are requests to render content on a list of virtual or physical devices.

Although the platform attempts to treat all devices in a homogeneous manner, their capabilities are used to distinguish the devices. The capabilities include simple parameters that configure a device. For example, FILE-NAME — the name of the file to be streamed, DESTINATION — the IP address and port number where the content is to be streamed. The capabilities also include PLAY, REWIND, FAST-FORWARD and STOP, which correspond to their respective tape deck control on a conventional audio/video player. Setting these capabilities is logically equivalent to pressing the corresponding button on the home entertainment center. These and other capabilities are unique to certain devices, and have different semantics on different devices. Compare, for example, a content source streaming a video and a collection of photos. For the video stream, PLAY, REWIND, and STOP have their familiar meaning. However, for a collection of photos, REWIND may mean go to the previous photo or go to the first photo in the collection.

Demo Setup The current demonstration of our prototype platform contains two content storage devices, three video rendering sinks and one audio sink. The remote control is implemented using Nokia 770 Internet Tablet, which runs Linux. The main control and processing hub is implemented on the BEE2 FPGA board [9], which contains five FPGA chips and has several functions in our demo. The main FPGA runs the Hub Server (a subset of the Ambient OS) on Linux. The main FPGA also serves as the router between the four slave FPGAs and external IP network. The slave FPGAs are configured at run time with trans-coders.

The demo has two small “human presence” sensors that communicate through sensor-network motes (running TinyOS [5]) to the Hub Server. They are implemented using floor pressure mats.

The demonstration highlights two important features of our system. (1) Automatic adaptation to a person moving around the house. This is simulated by a person moving from the first pressure mat to the second. The system relocates a running stream from one video rendering sink to the other, *i.e.* the video “follows” the person around the house. (2) The video rendering sinks accept video streams encoded in distinct formats. The first sink accepts only MPEG4 Level II Simple Profile, the other Motion JPEG. This does not present any problem for our system, since real time trans-coders are dynamically mapped into the implementation tasks.

Our demonstration highlights only several simple aspects of the platform we have developed and validates its key underpinnings.

5 RELATED WORK

Creating smart environments and enabling ubiquitous device interaction is an active research area. We consider related work that is at a higher level of abstraction and pursues goals similar to our work. First, [12] introduces a middle-ware abstraction for multi-media applications. ZUMA is similar to their architecture; both provide a higher-level view of the application (ZUMA's behavioral task) and refine the graph to an implementation. They also propose a uniform messaging system similar to the DRTS. An important difference is their focus on the integration of existing multi-media object protocols, so-called distributed object environments (DOE). Their abstraction lacks several concepts which ZUMA contains: current environment, personae/users, and handling of dynamics of a user-controlled environment aware system through the continuous constraint-based mapping strategy. Furthermore, they do not address the integration of devices beyond those used for multi-media, and seemingly underestimate the role of reconfigurable devices for handling the changes in multi-media encoding formats and communication standards.

Second, [11] has developed GaiaOS, an operating system for the home environment. Similar to ZUMA's environment, GaiaOS abstracts the physical space and devices into a unified system: an *active space*. Gaia also has a notion of high-level applications which are mapped onto the current active space. When users migrate between different active spaces their applications are dynamically re-mapped. The ZUMA platform is different from Gaia, in that we treat the content as a first class citizen (on par with devices and users). This enables migration of content, DRM, and a cleaner application abstraction that allows trans-coding of content.

Several other platforms exist [8], [10], [13]; they focus on abstracting devices in the environment and also connecting devices to each other. ZUMA goes beyond the simple abstraction of devices by including a notion of the multi-user optimal experience and optimization by migrating applications to different environments. The first-

class treatment of content further aids applications and allows on-the-fly trans-coding and reformatting.

The home environment is also targeted by private industry. The projects in the PC and consumer electronics sectors include Microsoft's mediaPC and research on Easyliving [7]; Sun Microsystems' Epsilon [1]. Further, the Universal Plug-n-play (UPnP) Forum [6] has been very successful in establishing standard device control protocols. The Digital Living Network Alliance (DLNA) [2] works to make existing standards (PC, CE, mobile) inter-operable. In the home automation space, Echelon established its LonTalk protocol [3] as a standard; companies like Smarthome [4] offer incompatible, proprietary products for lighting, heating and security systems. Companies like Intel have also been very active in the sensor network space. These commercial efforts focus mainly on a single segment of the home environment (multimedia, home automation, ambient sensor networks, or security), while the solution in this paper brings these aspects together.

6 CONCLUSION

We identified four key challenges for a smart-home environment platform: Zero-configuration, Universality, Multi-user optimality, and Adaptability (ZUMA). The proposed set of abstractions unifies users, physical devices, content in a single framework that enables extensibility, and readily admits heterogeneity in communication, operation, and encoding.

Zero-configuration enables easy device integration and functionality re-use, supported by our capabilities abstraction. Universality guarantees that every device can interoperate with another. Our distributed run-time system enables this by the necessary indirection between the devices and high-level abstractions. Multi-user optimality captures personalization and ease of use. These are enabled by the personae, content, and session abstractions. Adaptability requires the system to self-configure based on the movement of people, their requests, and changes in the environment. Sessions and the run-time continuous mapper are also key mechanisms to support adaptability.

We have implemented a prototype to demonstrate a subset of these abstractions and platform. The prototype has demonstrated key aspects of content, capabilities, and Ambient OS (run-time continuous mapper and distributed run-time system).

7 ACKNOWLEDGEMENTS

We wish to acknowledge our colleagues that helped to realize the work in this paper: Andrew Schultz, Alex Krasnov, Kaushik Ravindran, Stanley Chen, Dan Burke, Ken Lutz, and Subra Subrahmanyam. Thanks to Jan Newmarch for feedback on early versions of the paper.

REFERENCES

- [1] <http://research.sun.com/projects/epsilon>.
- [2] <http://www.dlna.org/>.
- [3] <http://www.echelon.com/>.
- [4] <http://www.smarthome.com/>.
- [5] <http://www.tinyos.net/>.
- [6] <http://www.upnp.org/>.
- [7] Barry Brumitt and Brian Meyers and John Krumm and Amanda Kern and Steven A. Shafer. Easyliving: Technologies for intelligent environments. In *HUC*, pages 12–29, 2000.
- [8] C. Jacquet, Y. Bourda and Y. Bellik. An Architecture for Ambient Computing. In *The IEE International Workshop on Intelligent Environments*. The Institution of Electrical Engineers, June 2005.
- [9] Pierre-Yves Droz. Physical design and implementation of BEE2: A high end reconfigurable computer. Master's thesis, UC Berkeley, 2004.
- [10] J.S.Y. Chin and V. Callaghan and M. Colley and H. Hagraas and G. Clarke. Virtual appliances for pervasive computing: A deconstructionist, ontology base, programming-by-example approach. In *The IEE International Workshop on Intelligent Environments*. The Institution of Electrical Engineers, June 2005.
- [11] Manuel M. Román, Christopher Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell and Klara Nahrstedt. A middleware infrastructure for active spaces. *IEEE Pervasive Computing*, 1(4):74–83, 2002.
- [12] Marco Lohse and Philipp Slusallek. Middleware Support for Seamless Multimedia Home Entertainment for Mobile Users and Heterogeneous Environments. In *Proceedings of The 7th IASTED International Conference on Internet and Multimedia Systems and Applications (IMSA)*, pages 217–222. ACTA Press, 2003.
- [13] N. Drossos, C. Goumopoulos and A. Kameas. A conceptual model and the supporting middleware for composing ubiquitous computing applications. In *The IEE International Workshop on Intelligent Environments*. The Institution of Electrical Engineers, June 2005.
- [14] Jana van Greunen, YuryMarkovskiy, Chris R. Baker, Jan Rabaey, JohnWawrzynek, and AdamWolisz. A platform for smart home environments – the case for infrastructure. In *Proceedings of the 2nd International Conference on Intelligent Environments (IE '06) (to appear)*. The Institution of Electrical Engineers, July 2006.