XTream: An Open, Distributed Platform for Processing Personal Information Streams

Michael Duller Gustavo Alonso
Systems Group, ETH Zurich
Haldeneggsteig 4
8092 Zurich, Switzerland
{michael.duller, alonso}@inf.ethz.ch

ABSTRACT

Personal information processing is an important and difficult challenge due to the many constraints involved and the need for open, easy to use systems. In the XTream project, we envision an extensible runtime platform and programming model that support users in easily defining complex data stream processing chains that can be arbitrarily connected to the streams of other users. The platform is based on modularization and service orientation to provide extensibility and handle the dynamism immanent in personal information processing applications. In this short paper we outline the most important aspects of the project.

Categories and Subject Descriptors

H.1 [**Models and Principles**]: User/Machine Systems—*Human information processing*; H.2 [**Database Management**]: Systems—*Distributed Databases*

General Terms

Design, Management

Keywords

Data Streams, Personal Information, Distribution, Information System, Personalization, Modularization

1. INTRODUCTION

Advances in networking, computing, and digital devices have led to a proliferation of data sources and the possibility of having access to these data sources anywhere and anytime. Such scenarios raise many issues: the distribution and dissemination of the information, in-network data processing, localization and context dependencies, delivery to the end user, data sharing, and, in general, the architecture of such a collaborative, Internet-scale data processing and dissemination system.

In this setting, personal information poses the biggest challenges. First, the abundance of sources (people and devices) and the many people involved create a significant problem of composition, as the information is most interesting when it can be combined, forwarded, and composed in a flexible manner. Second, a great deal of this information has the characteristics of streaming data and is constantly flowing—which is part of its value (phone calls, calendar alarms, short messages, e-mails, notifications, etc.). Third, the data involved is heterogeneous in terms of kind, frequency, volume, availability, and relevance with many of these aspects being highly dependent on user and context.

Today, there is no obvious complete solution to this problem. The most advanced support comes from the technologies referred to as Web 2.0, which aid people in publishing, exchanging, and processing some of their personal information. However, Web 2.0 technologies are typically restricted to centralized infrastructure (server farms), require people to "give away" their data, and are quite limited in terms of programmability and extensibility. The alternative to giving the data away is to install an engine such as, e.g., a data stream processing engine [2, 1, 6, 3]. However, these engines are big, expensive to acquire and/or maintain, not trivial to setup and extend, not easily composable, and not programmable by everyday users.

Our vision for XTream is that of an open and extensible platform that enables everyday users to effectively and easily process their personal information data streams and exchange them with families, friends, and colleagues around the world. In what follows we outline this vision and provide some details of the system developed so far.

2. THE XTREAM VISION

In XTream, we aim at developing a system based on modularization and service oriented design [8, 10]. XTream incorporates modern data processing techniques like stream processing into its model to provide flexibility, extensibility, interoperability, and rich programmability in a distributed and federated environment.

Figure 1 illustrates the data processing model that is at the heart of XTream. It consist of a mesh of adapters and processing elements (*slets*, short for stream-lets) connected by *channels*. On the left, data sources are adapted by α-*slets*. Data is then forwarded through a mesh of π-*slets* (processing slets), which are connected by channels. Channels buffer and forward data. On the right, data leaves XTream through ω-*slets*, which adapt data sinks.

As this processing model (correctly) resembles that of traditional stream processing engines, it is important to look into the applications XTream targets to better understand

WDDDM '09, March 31, 2009, Nuremberg, Germany

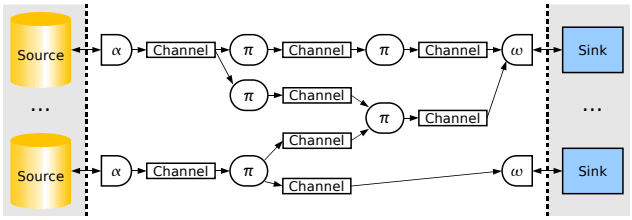

Figure 1: Data processing model of XTream

what makes XTream unique. XTream can be seen from three different perspectives: the end user, communities of users, and developers.

2.1 End User

For each single user, XTream is intended as a platform for managing information. XTream turns arbitrary sources of data into data streams and pushes them through the mesh of processing stages to deliver them to the user at, potentially, several end devices. XTream is intended to support multiple sources and multiple sinks for the information flow, with the sinks being in most cases different end devices (e.g., a computer, a screen, a PDA, a mobile phone). The sources of information we are considering include voice and text messages, reports on calls/SMS/e-mails from several addresses, RSS feeds, e-mail, photos from a digital camera, etc. XTream allows the user to define the sources, define the sinks, and specify from a library of processing steps how the data will be combined/filtered/processed as it moves from the sources to the sinks. XTream supports rules to decide which path the information should follow (e.g., depending on time of day, content, or user settings). Finally, XTream also supports the storage and caching of the data, offering functionality for querying the data in different ways.

2.2 Communities of Users

For communities of users, XTream offers the possibility of linking the *personal data processing and dissemination mesh* of each user with those of other users, thereby building an even larger mesh. Through standard interfaces in the processing stages and the communication channels between processing stages, XTream gives users the option of publishing any of the final or intermediate results of their personal processing mesh while other users can connect to that information and feed it into their own meshes. The scenarios that we are targeting include forwarding of notifications to other persons depending on conditions (e.g., user sets a *do-not-disturb* flag, routing to other users or devices depending on the time of day), raising and propagating alarms, content based routing, etc. In this way, XTream can be used not only to disseminate data among users but also to build data dissemination and processing meshes that are shared by a group of users (e.g., members of the same research group) and that directly feed on the personal meshes of each user. Taking to the extreme, XTream meshes could all be interconnected, in the same way that following a small number of links allows to reach almost any web page in the Internet. XTream has been designed to function and operate at such Internet scale.

2.3 Developers

For developers, XTream provides a very clean and rigorous system design. Processing happens in so called *slets*, which use standard interfaces for input and output and are language and OS independent. That way, the libraries of specialized (e.g., e-mail or SMS filters) or general (e.g., splitters, routers, duplicators) processing steps can be built independently of how they are combined by users. Communication between the slets happens through channels. Channels are strongly typed and support all the data management in the system. Slets put data into channels or read data from them. Channels store/buffer/forward the data from the slets providing the input to the slets interested in their output. There is no processing and no side effects in the channels, which treat data as push, pull, or both. Channels also support callbacks to send requests back along the reverse path of the processing and dissemination mesh. This allows slets to request specific data from given sources in either push or pull mode, regardless of whether the source supports push or pull. The architecture of XTream has also been conceived as a modular system where slets and channels can be added or removed at runtime, with XTream dealing with the corresponding dynamic changes. Thus, XTream has been designed to allow separate development of slets, of channels, and of the processing and dissemination meshes—which we envision may in fact be done by completely different people.

The separation of concerns between elements of the system, the clean interfaces between components, and the architecture of the system all contribute to simplifying the task for the end user for whom using XTream should be no more difficult than using a web browser.

3. DATA PROCESSING MODEL

Information processing happens in XTream in a mesh of slets and channels, as outlined in Section 2 and illustrated by Figure 1. In this mesh, data items—discrete units of information—flow from sources on the left to sinks on the right and are processed by π-slets in between. Data items can be pushed from an slet to the downstream channel and then either pushed further by the channel to the connected slets or buffered for later access by the connected slets, depending on their requirements. Channels thus allow downstream slets to pull data from upstream push slets by decoupling them with a buffer. Furthermore, pulling slets can specify the buffer window the channel should keep for them, e.g., the ten most recent items or all items that arrived in the last five minutes. Information that arrives in a push manner and thus by virtue of the upstream processing mesh (regardless of whether it is further accessed using push or pull) is termed to be on the *stream data path*.

In addition to processing fresh data arriving on the stream data path, many applications on personal information also include access to existing data, like e-mail messages that are stored in the inbox or archived photos. To access this data on the original sources or a channel on the upstream processing mesh that buffered them, XTream provides the *query data path*. This path allows downstream slets to ask the upstream channel they are connected to for all items that it logically contains. A channel can be seen as a view representing the upstream mesh it is connected to. If the channel materialized all items, it simply returns them. If not, it will ask the upstream slets to return all items which causes them to fetch items from their upstream channels, process (e.g., filter) them, and return the result. As an optimization, the slet that initially requests all items can instead send a query

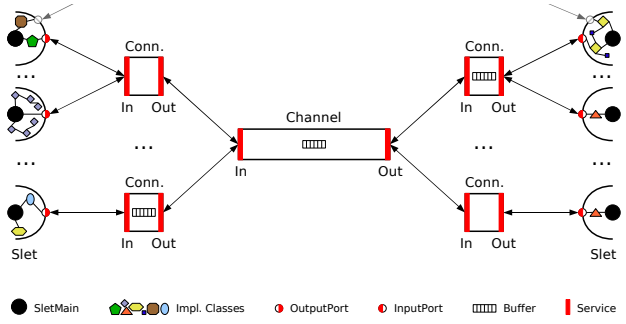

Figure 2: Execution and implementation model

to the channel that corresponds partially or fully with the requesting slet's processing. If slets in the upstream mesh understand the query, it can be used to reduce the number of results sent to the requesting slet—pushing selectivity towards the data sources.

4. EXECUTION MODEL

XTream is implemented using a Service Oriented Architecture (SOA). Components like slets and channels interact with each other through well defined service interfaces. Figure 2 illustrates the execution and implementation model of XTream by depicting all implementation details, from slets emitting items on the left, to a channel, to slets consuming items on the right. Note that, compared to the higher level data processing model, *connectors* have been added as another class of components. Furthermore, the internal implementation of slets, their input and output ports, as well as buffers have been made explicit in the illustration. Arrows between components are bidirectional as they represent the component interaction in terms of service method invocations rather than in terms of data flow. Ellipses between slets or connectors indicate that any number of instances thereof can exist and interact with one instance of a connector or a channel, respectively. For the sake of readability, only service interfaces used in data exchange have been depicted using thick, red bars. Service interfaces for managing components are not explicitly depicted but the component as a whole represents the respective service.

In the following sections, we give insight into some important details of the execution and implementation model.

4.1 Slets

π-slets have any number of input and output ports. They can create and remove ports during initialization or at runtime. Individual properties can be assigned to every instance of an slet. These properties, a set of key/value pairs, customize the behavior of the particular instance.

A π-slet can become active under three circumstances. The first case is the arrival of new data on the stream data path of a channel connected to its input port; this triggers, corresponding to the requirements that the slet stated, the immediate push of items to or notification of the input port of the π-slet. The second case is the arrival of a request on the query data path from a channel connected to one of the slet's output ports. The third case is the updating of properties at runtime. When an slet becomes active under one of the first two circumstances, it can always access both data from the stream data path and data from the query data path. This capability bridges both access variants to data and thus is the key to full integration of streaming, non-streaming, and hybrid data sources as well as push and pull processing.

Every port of an slet offers an *InputPort* or *OutputPort* service providing methods for pushing data to the port or querying it, respectively. Ports expose their requirements with respect to buffer policy and access method in their service properties.

The processing model for α- and ω-slets is very similar to that of π-slets, with the exception that α-slets cannot have input ports and ω-slets cannot have output ports.

4.2 Channels

Channels are provided by XTream. Depending on the requirements of all the connected slets, and whether the channel is operating in materialized mode or not an implementation will be chosen that efficiently satisfies these requirements. The implementation can change at runtime to continuously meet the requirements posed.

Channels expose a *Channel* service interface for management. Additionally, every channel also exposes a *ChannelInput* service interface used by input connectors and a *ChannelOuput* service interface used by output connectors. These interfaces provide methods for data exchange between connectors and channels. Furthermore, channels also form the well-defined interface for remote data exchange and every channel has a unique URI.

4.3 Connectors

Connectors provide a level of indirection in the interaction between channels and slets. Slets' ports connect to connectors instead of directly to channels. Connectors for channel inputs expose a *CICI* and a *CICO* service interface (Channel Input Connector In and Out) and connectors for channel outputs a *COCI* and a *COCO* service interface (Channel Output Connector In and Out).

In a local, non-optimized setting, connectors simply pass through requests between channels and ports. They collect the buffer requirements of all connected slets and communicate them to the channel. In a distributed setting, connectors provide means for accessing a channel in a remote framework as well as for implementing optimizations like, e.g., local caches, where the connector itself can fulfill the connected slets' buffer requirements.

4.4 Distribution

In a distributed setting, interaction between instances of XTream happens in a peer-to-peer manner. Access to channels on a remote framework is realized using *remote connectors*. One half of a remote connector is installed in the framework where the channel resides and implements either the *CICO* or *COCI* service interface. The other half of the connector is installed in the framework where the channel needs to be accessed and implements either the *CICI* or the *COCO* service interface. The two parts communicate with each other and act as one logical connector. Typically, the latter part implements a local buffer that satisfies the buffer requirements of all the connected slets. Using connectors as local proxies for channels allows to, i.a., move the buffer for an slet to the node on which the slet runs and thus reduce access times and save bandwidth. It also allows for offline operation as the transition between online and offline hap-

pens inside the connector while the composition of slets and channels with the connector can be left unchanged. Hence, remote connectors are also a good example for the ease of extensibility of the system due to the well defined service interfaces.

5. PROTOTYPE

We have implemented a first prototype of the XTream platform taking advantage of the module management features of OSGi [7]. OSGi is a dynamic module and service platform for Java used in a number of products from the Eclipse IDE to embedded software in cars. We have implemented two applications on top of our prototype to test our ideas and demonstrate their feasibility.

5.1 Personal Information on PlanetLab

To show XTream's ability to process and disseminate information at Internet scale, we selected 200 nodes of PlanetLab [9] at different sites and deployed a synthetic application on each node. The application simulates users that consume data streams, process them, and publish some of them to be consumed by other users—an abstract description of the collaborative, personal information processing applications we have in mind. The result is a global data stream processing system implementing a complex and dynamic processing mesh of slets and channels. Every node operates autonomously and the experiment exposes our platform to dynamic and unpredictable situations, namely the setup phase of the processing mesh and inevitable node churn in PlanetLab.

After 24h hours of running we analyzed the data logged on each of the 200 nodes and found that XTream had operated successfully in a highly distributed manner, supporting a large scale processing and dissemination mesh where nodes connect spontaneously to stream sources and publish new streams in a continuous manner. The experiment also demonstrated that the depth (or length) of the processing pipeline is not an issue for XTream, which can sustain processing chains of arbitrary length. Another aspect explored in the experiment was the ability of channels to deal with varying amounts of data arriving concurrently. We did not perceive any significant impact on average transmission times between nodes during the entire length of the experiment. Finally, XTream proved to be resilient to failures of individual nodes, an advantage of how component interaction is implemented in XTream. This is exactly the behavior needed to implement large scale, collaborative exchange of data streams where failures and changes are automatically masked form the user.

5.2 Linear Road Benchmark

To show XTream's extensibility and low overhead, we have implemented the Linear Road Benchmark [4] on XTream. We ported the implementation presented in [5] by wrapping the XQuery engine into an slet, wrapping the storage implementations into channels, and then creating the same workflow that was used in the paper using these new slets and channels. The XTream implementation performed only marginally worse and was able to sustain the same load factor as the original implementation. Also the memory consumption throughout the benchmark was only marginally higher than without XTream. Porting the Linear Road Benchmark to XTream demonstrated its ability to encompass any form of data processing and its extensibility. The benchmark results gave a clear indication that XTream's design induces very little overhead in terms of memory and processing overhead, despite the dynamic, service-based interaction.

6. CONCLUSION

Processing of personal information raises significant challenges due to its dynamism and heterogeneity. The fact that such information is most valuable when it is composable also creates a difficult design problem. In this short paper we have presented XTream, our prototype of an extensible platform implementing large scale data stream processing meshes carrying personal information. XTream allows people to individually combine, process, and publish their personal information while still being extensible and providing potential for optimizing these applications.

7. REFERENCES

[1] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Çetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. S. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik. The Design of the Borealis Stream Processing Engine. In *CIDR*, 2005.

[2] D. J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: A new model and architecture for data stream management. *The VLDB Journal*, 2003.

[3] A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R. Motwani, U. Srivastava, and J. Widom. STREAM: The Stanford Data Stream Management System, 2004.

[4] A. Arasu, M. Cherniack, E. F. Galvez, D. Maier, A. Maskey, E. Ryvkina, M. Stonebraker, and R. Tibbetts. Linear Road: A Stream Data Management Benchmark. In *VLDB*, 2004.

[5] I. Botan, P. M. Fischer, D. Florescu, D. Kossmann, T. Kraska, and R. Tamosevicius. Extending XQuery with Window Functions. In *VLDB*, 2007.

[6] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. Shah. TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. In *CIDR*, 2003.

[7] OSGi Alliance. OSGi Service Platform. `http://www.osgi.org/`.

[8] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Commun. ACM*, 15(12), 1972.

[9] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A blueprint for introducing disruptive technology into the Internet. *SIGCOMM Comput. Commun. Rev.*, 2003.

[10] W. P. Stevens, G. J. Myers, and L. L. Constantine. Structured design. *IBM Systems Journal*, 13(2):115–139, 1974.