

INDUS: A NEW PLATFORM FOR UBIQUITOUS COMPUTING

Kallol Borah

Indus is a software agent platform for ubiquitous computing. Ubiquitous computing is a term used to generally refer to computing across software platforms and hardware devices to seamlessly interface human to machine and machine to machine. Work on platforms for ubiquitous computing have been continuing throughout the past decade in academic and industry research organizations. The Indus project was conceptualized in 2002 and prototypes implemented at the Indian Institute of Technology Madras and the Indian Institute of Science Bangalore to demonstrate how general purpose object oriented programming languages can be extended to enable ubiquitous computing applications. The project is sponsored by Aumega Networks, a company focused on development of software infrastructure for ubiquitous computing and is run in consultation with several academic research institutions.

The primary components of the Indus platform comprise of a programming language to implement software agents, libraries to provide services to agents on a distributed network and containers or run time environments to enable deployment of agents on a variety of hardware platforms starting from 8 bit devices onwards. In Indus, software agents represent language abstractions that are autonomic, adapt to existing computing environments and coordinate with other agents to cooperatively execute tasks.

The Indus programming language enables implementation of software agents that have the ability to coordinate with other agents and compose components using connectors. Connectors are language abstractions that implement the logic of binding components programmatically.

The libraries for the Indus programming language are segmented according to classes of target host platforms : enterprise platforms, mobile hosts and motes. Motes is a term used to refer to a hardware platform, often wireless, combining a radio front end with a micro-controller and a sensor back end. Standard Indus libraries provide services to deployed agents and components such as activation strategies, lifecycle management, service discovery, routing, communications, interoperability, transaction management, persistence and security. Indus Enterprise libraries are to be specified to enable provision of grid services to agents deployed on enterprise platforms. Indus Micro libraries are to be specified to enable provision of services to agents deployed on mobile hosts. Indus Mote libraries are to be specified to enable provision of services such as timer, power and interrupt management, etc to agents hosted on mote platforms.

Indus applications compile to Java byte code that can be interpreted by a Java Virtual Machine (JVM) and is suitable for execution on enterprise platforms. In case of mobile hosts, Indus compiles to C code and binary executables that are managed by the Indus Micro run time. In the case of motes, the Indus compiler generates code targeted at the device platform.

Indus can thus be thought of as a common language front end for generating code that can execute concurrently on a variety of 8/16/32/64 bit platforms and can seamlessly communicate with each other to implement a variety of pervasive computing applications.

1 THE INDUS PLATFORM

Existing middleware for client server computing such as .NET, J2EE and Corba were not designed for ubiquitous computing. .NET is the Microsoft initiative for component programming on the Windows operating system. Java 2 Enterprise Edition abbreviated as J2EE represents Sun Microsystems' effort to advance Java as a platform for distributed computing. Corba is the Object Management Group's (OMG) initiative for component centric computing. Indus aims to meet the requirements of adaptive middleware for ad hoc, device centric computing networks. The foundation of the Indus platform is an object-oriented language Indus - for implementing software agents.

The ubiquitous application developer is therefore expected to model any system as a set of concurrently executing agents, with each agent representing a role and then use features of Indus new types such as agent and component, component ports, syntax and semantics for composition and coordination, new libraries for distribution - to implement and deploy agents across a variety of peers' (enterprise servers, mobile terminals and motes).

2 THE INDUS PROGRAMMING LANGUAGE

Why is a new programming language required for ubiquitous, adaptive computing ? That is often the first question that needs answering. For client-server applications that execute on networks with mostly static topology, code for aspects such as synchronization, concurrency, parallelism and distribution are all written together with business logic, which leads to tangled code. Indus, by design, model application systems as concurrently executing processes (agents) that compose components (reusable static code) and coordinate with each other to execute application functionality over a network.

The Indus programming language features syntax and semantics that are similar to Java but provides additional features such as

- **New reference types:** Indus does away with the Java type called Interface and instead substitutes it with two interface types called Agent and Component. Every Indus class must therefore implement an Agent or Component.
- **Coordination styles:** All object oriented languages support only the call and return' style of interaction between clients and servers where clients call servers that process logic and return calls to clients. Indus defines syntax and semantics for several coordination styles, e.g., parallel pipes, voter-coordination, blackboards, polling, broadcasts, callbacks, etc, to facilitate a variety of interactions between concurrent processes (agents). Coordination styles separate aspects of concurrency and communication from distributed programming by abstracting out logic for cooperative execution by concurrently executing programs, for example, in the voter-coordinator style of coordination, an agent (representing the coordinator) asks other agents to 'vote' on a task and only if all participating agents vote in the affirmative is the task committed to and executed by the coordinator. Similarly, in the agenda style of coordination, a token which is often a data structure representing a problem is rotated between agents that are

allowed to read and write to the token till an answer is found. Indus has thus brought coordination abstractions to message passing programming models of distributed computing.

- **Composition styles:** Indus defines syntax and semantics for several composition styles, e.g., pipes and filters, events, rules, etc, to enable plugging of components by agents. Component composition facilitates code reuse in Indus as agents that are active objects compose reusable static objects or components programmatically. An Indus developer is expected to use connectors represented in the form of operators in Indus to programmatically compose components.
- **Ports:** Indus defines a type called Ports that define the ability of components to be plugged in a number of ways. Ports validate component compositions.
- **Behavior inheritance:** Indus places more importance on interface composition as a way of inheriting behavior than stress on class-based inheritance.

3 INDUS LIBRARIES AND SERVICES

In Indus, the code for synchronization and management of concurrent agents has thus been subsumed within the language itself whereas the code for managing distributed agents is provided by the library support to the Indus programming language. These are in addition to the usual Java-like language library support available for math, graphics, utilities, etc.

The Indus Standard libraries provide general libraries available freely to every agent application developer. The Indus Standard libraries provide services for

- **Self configuration:** this enables an agent or component to publish services provided and set parameters such as peer ids, concurrency and transactional policies, and its implementation type. By implementation type, software agents can be classified as User agents (identified by a role that can be played by a number of agents executing on different threads of execution) or Process agents (identified by a role that can be played by agents sharing a single thread of execution); both types of agents are transactional, history aware and persistent. By implementation type, software components can be classified as Service components (non-transactional, not history aware) and Session components (non-transactional and not persistent but history aware within a session boundary).
- **Lifecycle management:** this comprises abilities to deploy agents and components and evaluate implementation types and associated invocation rules of agents and components prior to their invocation.
- **Activation strategies (proactive and reactive):** this comprises facilities to queue client requests and schedule them according to different activation strategies. Proactive activation would result in a first in/first out (FIFO) schedule whereas Reactive activation would schedule agent invocations as events occur.

- **Service discovery:** libraries for service discovery enable every channel and connector (these are communication abstractions in Indus hidden from the Indus programmer) to discover invoked services on local and remote hosts.
- **Communications:** libraries enable run time binding to adapters of various types. This enables a high level of interoperability between agents implemented in Indus and components implemented in other languages such as Java and C++.
- **Intelligent routing (route discovery, maintenance, shortening, salvaging):** this comprise of facilities to register and exchange route information between routers running on every agent host, route groups that allow maintenance of routes, discovery of least hop routes and route salvaging.
- **Transaction management (includes nested transaction management):** facilities comprise formation of transaction boundaries across concurrent processes (agents), exchange of transaction data across transaction managers running on agent hosts and commit/rollback operations on transactions.
- **Persistence:** library support for persistence enables mapping of agent parameters to data stores and their retrieval when required.
- **Policy management:** this includes support for attaching policies at run time to agents and components, and mechanisms for their evaluation at run time.
- **Security:** library support for security comprises of different encryption schemes for location data, messages and mobile agent code.

4 INDUS RUN TIME ENVIRONMENTS

The Indus compiler family enables step wise but seamless compilation of Indus code to byte code, byte code to C code, and finally to machine dependent executable code.

Indus implementations can therefore execute across a variety of enterprise, mobile and mote platforms from a large number of vendors. On enterprise platforms, Indus agents and components compiled into byte code are managed by the Indus Enterprise container that executes on the Java Virtual Machine(JVM) and is therefore portable to a variety of underlying Operating System (OS) platforms. On mobile platforms, Indus agents and components compiled into C executables are managed by the Indus Micro container that is portable across a number of mobile Operating Systems. On mote platforms, usually 8-16 bit devices, Indus agents and components are managed by the Indus Motes container that is portable across a number of underlying core platform architectures, e.g., the 8051.

Indus run time capabilities varies across platforms. In cases where Indus compiles into Java bytecode, the run time capabilities are provided by the Virtual Machine. On embedded 16/32 bit platforms where Indus translates to C and cross compiles to architectures like ARM, MIPS, SH,

PPC and i386, the Indus container provides additional features such as multi-tasking, inter-process communication, synchronization, memory management, access to native APIs (timers, interrupts, ports, etc) and message routing. The Indus Micro container can also be plugged to Real time Operating Systems and there is an available implementation of Indus using the eCos RTOS. For 8 bit platforms such as the 8051, the Indus Mote container provides everything that the Indus Micro container provides except for the garbage collector which is not required since memory is allocated on the stack.

5 APPLICATIONS AND BENEFITS

Interest in ubiquitous computing cut across a variety of application segments. The target application segments below have therefore been identified keeping in mind the maturity of markets, commitment of resources by suppliers and the market's propensity to adopt.

- Control & Automation: Homes, Buildings & Industry
- Personal Area Networking & Mobility
- Supply chain management
- Intelligent Transport Systems
- Patient care
- Utility management

The Indus platform has been engineered to provide the following benefits to adopters

- **Higher productivity:** Indus consistently shows a productivity benefit of 30% over Java, measured in lines of code, for concurrent applications. This is achieved by enabling coordination among concurrent processes (agents) and reuse of components.
- **Multi-platform deployment:** Indus is truly write once, run everywhere. Unlike Java, Indus compiles to native code on a variety of host operating platforms .
- **Zero configuration requirements:** Indus promotes self-configuration, self-organization and self-healing of a network. Programmers are not required to write any code for these capabilities.
- **Zero downtime:** Indus applications allow run time change in application behavior without requiring any downtime at run time. This is achieved through the run time discovery and run time binding capability of Indus agents and components.
- **Quality of Service (QoS) management:** The Indus platform is QoS aware. QoS awareness is reflected in routing services that communications abstractions in Indus like channels that are used by agents and connectors that are used both by agents and

components use. In addition, transaction management features are tightly coupled to the language itself making it possible to mark a part of a class implementation as a transaction boundary that can also nest other transaction boundaries declared in the same agent or other agents. In Indus, various services such as routing, activation, security, resource management are designed to make them proactively manage QoS rather than reactively act to QoS issues.

6 AVAILABILITY AND FUTURE DIRECTIONS:

The Indus Standard platform is a basic and freely available version of the Indus platform. It comprises of the Indus programming language and basic run time libraries and is available freely from < <http://developer.aumeganetworks.com> >.

The immediate future of the Indus platform will focus on the launch of Indus Motes that will target a variety of 8/16/32-bit embedded platforms. The libraries and run time will remain open source to facilitate public contribution to the project. Indus Micro for mobile devices will also comprise of open source libraries and run time container.

Research is continuing to add more libraries to the Indus Standard platform to enable resource aware computing. Capability to monitor computing resources, schedule tasks according to resource availability, security, data management and a container console to configure the Indus container's run time services will be primary features of the extended version of Indus Standard.



Kallol Borah is the chief architect of the Indus platform. Kallol started the Indus project at the Indian Institute of Technology Madras in 2002 and continues to support Indus related research and implementation. He can be contacted at <mailto:k.borah@aumeganetworks.com>

References

Sinha, Navin. (June 2005). "The Indus Programming Guide", retrieved from http://developer.aumeganetworks.com/indusstddsk/beta.pkg/1.0.0/IndusProgrammingGuide_1.2.pdf

Borah, Kallol. (August 2005). "Indus: An Object Oriented language for Ubiquitous computing", to be published in Sigplan Notices <http://www.acm.org/sigs/sigplan/>

Wikipedia, July 2005. "Indus Programming Language", retrieved from http://en.wikipedia.org/wiki/Indus_programming_language

Andrews, Gregory R. (March 1991). "Paradigms for Process interactions in Distributed programs", ACM Computing Surveys, 29.1 pg 49-90