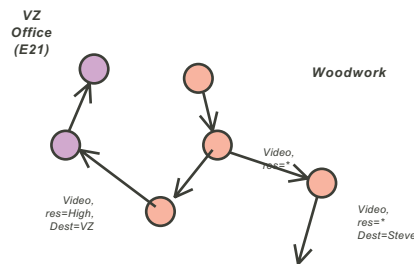


Pebbles: A Software Component System

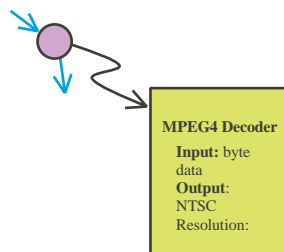
Steve Ward, Chris Terman, and Umar Saif

Introduction: The distributed, ubiquitous computing environment proposed by Oxygen and related projects demands a substrate in which each application can be viewed as a collection of pebbles rather than a single, monolithic boulder. The dynamics of the system, introduced by entities moving, failing, leaving and joining the system, suggests that services cannot be assembled at compile-time from a fixed set of components. Instead the system needs to be able to discover the most suitable components currently available in the system and dynamically compose and adapt services as the system evolves. In order to reconcile the dynamic nature of the system, including the introduction of new components, with nonstop service, the component "pebbles" must be individually replaceable. In order to facilitate automatic component assembly (e.g., to react to dynamic changes in the manner addressed by the related goals project) the pebble's function needs to be described at a sufficiently abstract level that candidate replacements can be mechanically detected.

Our goal is to provide a platform-independent substrate that allows a set of software modules to be selected from a repertoire, deployed on various hosts, and interconnected to implement some application-level service. We would like this system to allow hot swapping of code modules to upgrade service, as well as controlled but independent evolution of individual code modules ("pebbles").



Approach: We propose to build a middleware layer that provides support for locating the most suitable components in the system, deploying components at various hosts, and interconnecting components to dynamically compose and adapt application-level services. Further, we propose to build a library of pebbles, each associated with a unique identifier that establishes (at some abstract level) its function. Pebbles are in general platform independent, and hence there may be multiple implementations of each pebble. Each pebble corresponds to a particular function at a detailed



enough level that, in theory, the specification of a pebble as an application component is sufficient to guarantee the acceptable operation of that application.

In order to standardize the characteristics of each pebble, we use as the pebble's unique identifier a URI identifying an immutable description of that pebble. The description will, in general, contain a mix of formal interface specifications (method signatures, etc.), informal descriptions (of the sort found in man pages), and arbitrary other potentially useful information including code for test cases and demonstrations. This target description, encoded in XML, will serve a role analogous to that of WSDL descriptions for web services, and may use similar mechanism.

Pebbles thus provide for an intermediate description to serve as an interface between the application requirement and an actual implementation. They serve to commoditize services provided by code modules, allowing (say) alternative implementations to compete for adoption in an application in much the same way that commoditization of DRAM specifications allows vendors to compete for existing sockets.

Progress: This is a new project.

Future: Plans for an early prototype implementation involve a primitive bootstrap listener on each host that mediates requests for pebbles by loading and caching code locally from a library associated with each type of host.

Longer-term implementation plans involve support for compilation during the installation process. Using this approach, instances of pebbles can be compiled to exploit instance-specific peculiarities; for example, connected pebbles within the same host might be recompiled to intercommunicate directly and to share data structures.