

ROBOCOP

Robust Open Component Based Software
architecture for Configurable Devices Project

- Project of some European Companies
- Small and Large Companies
- Private and Public
- Define a development model

Author : Thomas Seppi

Aim

- Define an open component-based partial architecture for the middleware layer in high volume appliance
 - Enable robust and reliable operations with components
 - Introduce the possibility to upgrade, extend or trade components
-
-

Scope

- Development Framework
 - It defines a number of aspects of the development, trading, and downloading of Robocop Components.
 - How Components are developed (methodology, tools, languages, etc.) is outside the Robocop scope.
 - Published Components may be “generic” in the sense that they still need to be tailored to run on a specific Platform
 - Execution Framework
 - Defines a Component Model for the Middleware layer of an unspecified, potentially connected, single device.
-
-

Architecture considerations

Trade-off between the requirements, concerns and constraints, according to these guidelines:

- Simplicity
- Minimizing assumptions
 - OS features, Hardware etc.
- Favour open over closer solutions
- No penalties for non-used features
- Implementation preference: Run-time, servers and clients



Assumptions:

- Minimal platform functionality
 - Scheduling
 - Process concept
 - Run-time activation
 - Persistence storage
 - Platform standards: For a given platform:
 - Standard representation for object, library and ex. Files
 - Standard mapping from high-level languages to the processor
 - Hosts connected to the Internet where needed
-
-

Entities

- Computer systems
- Component
- Run-Time Environments
- UUID
 - Is the Universal Unique Identifier, a 128 bit number that identifies a component.



Entities (1) - Computer Systems

- Hosts
 - Computer systems where components are developed, certified, trade, tailored and integrated
 - Repository
 - host where components are published, trading and downloading functions
 - Device
 - Computer systems that hosts and execute the Software (Applications, Components, The Robocop Run-Time Environment and the OS)
-
-

Entities (2) - Component

A component in Robocop consists of a set of models:

- Model: is anything that conveys informations
 - Resource Model
 - Simulation model
 - Documentation
 - Functional model
 - Source code
 - ...
 - Executable model



Entities (2) – Component (2)

- Executable model: a special model that contain the executable code and all the informations
 - Manifest: is the “table of contents” of a component
 - It list all the available models and their relations
 - Package: the physical representation of a component
 - Service: is the functionality offered by a component
 - Instance: like an object in OO programming
 - Manager: like “new” in OOP
 - Interface: like an abstract class, it can be extended (not multi-inheritance) and it's language independent
 - Reference: point to a Platform specific standard binary constructs like data and operations.
-
-

Entities (3) Run-Time Environment

Is the Robocop component framework on a device.
It is platform dependent.

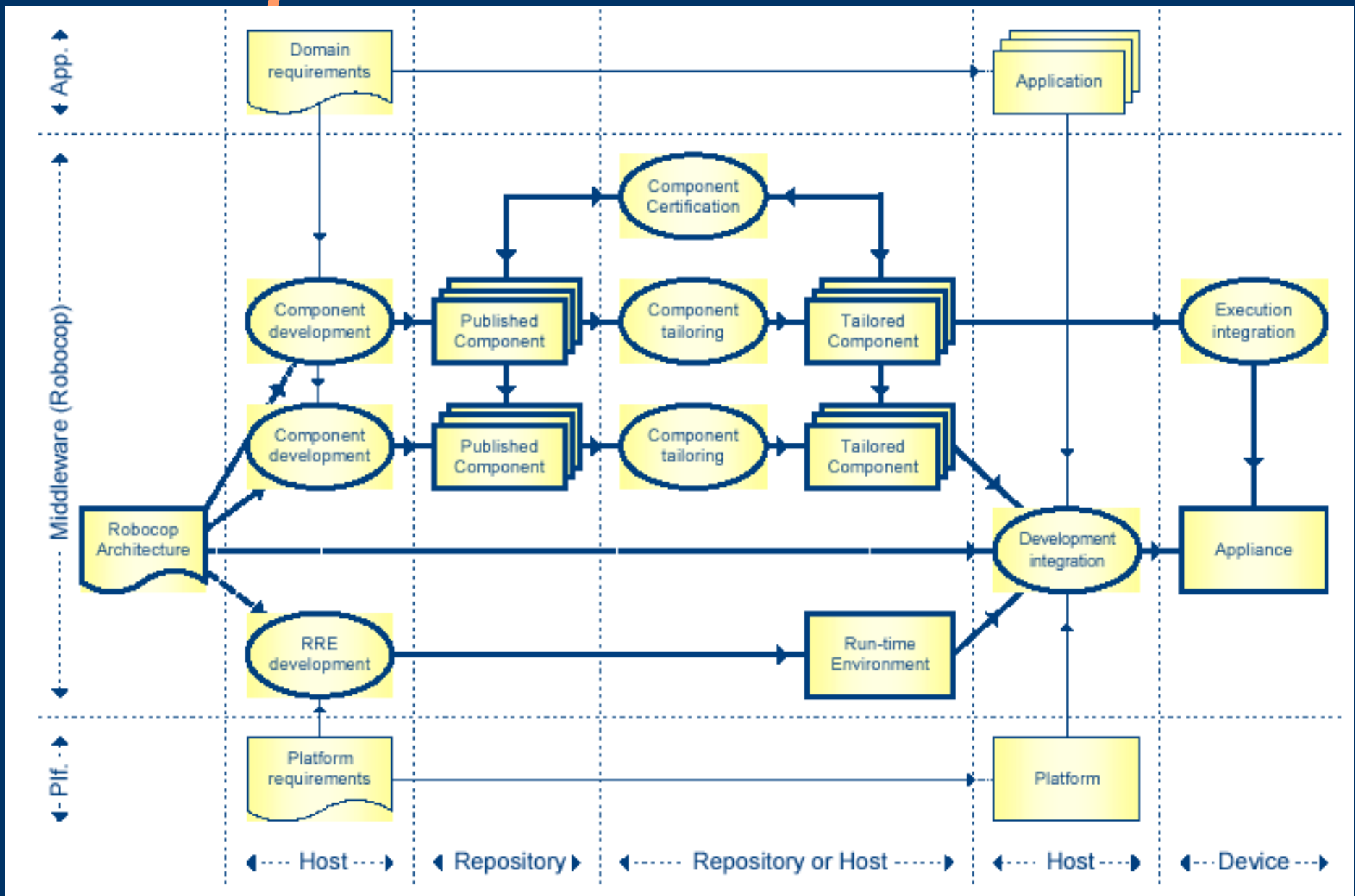
- **Component support:** Implements the Robocop component model on a given platform.
 - **Registry:** refers to the persistent storage device
 - **Download support:** provides the capability to register new components in a device
 - **OS abstraction layer:** provides OS functions like mutex, memory, management, timers ect.
-
-

Development framework

The development framework defines the roles and the relations between the various entities involved in the development, certification, trading, tailoring, and integration of Robocop Components and Robocop Runtime Environments.



Development flow



Development model

- Component Development
 - Component builders develop components according to application requirements.
 - Repositories
 - Store components, support searching, uploading and downloading.
 - Certification
 - Independent validation that a Component adheres to the Robocop architecture and/or that it meets certain quality attributes.
 - Increase the trusts in a Component
-
-

Development model (2)

- Tailoring
 - Is the activity of specializing a Component to be deployed on a specific Platform.
 - may also include adaptation and/or creation of other Models
 - does not fundamentally change the Component
 - Development Integrations
 - covers all activities required to take existing Applications, Components and a Platform and make it all work together as a whole.
-
-

Development model (3)

- Execution Integration
 - New Components may become available that can replace existing Components to improve operation or may be added to create new capabilities
 - These Components can be loaded in the Device and registered with the Robocop Runtime Environment so that they become available to the Appliance.



Use of Models

Specifying a Component as a set of Models is a key innovation in Robocop.

- Support trading
 - evaluation versions of Components can be created by including the specification Models or by providing trading versions of (Executable) Models with certain limitations.
 - the existence of certain Models and/or their content can be used as search criteria



Use of Models (2)

- Support composition
 - Specifying a Robocop Component as a set of interrelated Models supports the reasoning on compositions of Components based on the composition of the Models.
 - Example: To determine the total size of the executable code segment of all Components, one can easily add the sizes of the individual Component's executable code segments.
 - Example: Given a behavioural simulation Model of a number of Components the behaviour of the composition of these Components could be simulated.
-
-

Use of Models (3)

- Support Execution-time Inspection
 - At execution time Applications, other Components, and the Robocop Runtime Environment may inspect Models to determine certain attributes of a Component or Service. Based upon this inspection different actions may be taken.
 - Example: During the creation of a Service Instance, the RRE will inspect the Resource Model(s) of the Service to determine whether the Appliance still has sufficient resources (RAM, CPU cycles, etc.) to host the Service Instance.
-
-

Programming model

The programming model offered by the Robocop architecture is based upon objects and interfaces. The objects themselves are opaque in the sense that it is only possible to interact with the object through interfaces. Interfaces to objects can be obtained by creating Service instances or can be returned by functions or operations.



Interfaces to Objects

- An interface definition specifies the operations supported by the interface and the semantics of those operations, and is purely functional.
- At run-time, an interface instance refers to an implementation of the operations of the associated interface definitions and to the data upon which the operations operate.



Interfaces to Objects (2)

- Inheritance
 - Interfaces can only inherit directly from one other interface: single inheritance.
 - It is allowed to have an inheritance chain, but not cycles.
 - Interface Navigation (“casting” in OOP)
 - When an object can be accessed via multiple interface instances, these instances are of a different type.
 - *QueryInterface* is the operation to navigate through interfaces, accepts an IID and returns an interface reference
-
-

Interfaces to Objects (3)

- Object Lifetime
 - It is not feasible to give the clients complete control over the lifetime of the object.
 - Two functions are part of each interface: *AddRef* and *Release*.
 - When the number of *Release* calls matches the number of *AddRef* calls, there are no longer clients using the interface instance.
 - When none of the interface instances on an object are in use, the object is no longer in use. The object can thus be destroyed.
-
-

Services

A Service provides a higher level of programming. It is a special kind of object that provides an interface of type *IService*.

- Provides Interfaces
 - A provides interface is an interface instance that can be accessed through an instance name. These instance names must be unique within the scope of the Service.



Services (2)

- Requires Interfaces
 - Service Instances can depend upon functionality offered elsewhere in the system. This dependency is expressed by requires interfaces. A requires interface is a binding point for interfaces.
 - Properties
 - Are named values that have a meaning for the service. These can be used as an alternative to Set/Get functions in a service specific provides interface.
-
-

Services (3)

- Service Manager
 - For all Service Instances of a given type obtained from a component, there is one Service Manager.
 - An important responsibility of the Service Manager is in the creation of Service Instances.
 - Static Services
 - The Service Manager allows only one instance.
 - Once the Service Instance has been retrieved, setting defaults (properties and requires interfaces) in the Service Manager has no effect.
-
-

Services vs Components

- The RRE determines the appropriate executable component when a Service Instance is requested.
 - It will do that autonomously based upon information received during registering of the component.
 - Different RRE might have a different policy to determine the most appropriate executable component for the requested Service Instance.
-
-