The Palladio Component Meta Model: Towards an Engineering Approach to Software Architecture Design

Ralf Reussner, Steffen Becker, and Jens Happe

Abstract— **The abstract (100-200 Words).**

Index Terms— **Component, Quality of Service**

I. Introduction

THIS is the introduction [**?**]. Components are for composition. Divide and Conquer. Provide a hierarchical structure of a software system. COTS failed. But Product-Lines work.

Szyperski's Component: - unit of independent deployment - unit of third party composition - has no (externally) observable state Our focus: Unit of independent deployment, all agree, but nobody knows what it is. - For us, it is more than copying a DLL.

For the Quality of Service prediction of a software architecture, we need a component model with clear semantics and enough information to conduct analyses. Current component models lack these capabilities.

II. Components in Software Engineering and Practice

What is a software component today? UML 2.0 (history, UML 1.0 Component = unit of deployment) - Encapsulate their contents - Are replaceable within its environment - Define behaviour in terms of provided and required interfaces

”As such a component serves as a type, whose conformance is defined by these provided and required interfaces.”

Corba Component Model J2EE, COM+

SOFA

Software Engineering Processes (RUP, speziell CBSE etc.)

Manuscript received January 20, 2002; revised August 13, 2002.
M. Shell is with the Georgia Institute of Technology.

QoS Prediction models (Trivedi, Balsamo, QoSA Reliability Survey Paper)

Cheesman and Daniels

So, what is a Software Component? o Realisation - Component = class(es) ? - How to bind provided and required interfaces? - How does the component interact with its environment? o Quality of Service - Assembly, deployment, internal structure o Runtime - How does the system look like at runtime? o Substitutability and Interoperability

III. Interfaces

Semantics related to Java and C# - free floating - protocols - type system - subtyping - conformance, - contra-variance

IV. Component Types

Is the type defined by its provided and required interfaces? - -¿ This is not true for every case. - Provided interfaces only? - -¿ This is not true for every case.

Distinguish between Provided and Complete types. Both have their advantages. They are defined by substitutability.

During software development components evolve, so there is no clear border between both types.

Different interpretations of required interfaces: - can be used, but there can be more (provided type interpretation) - can only be used (complete type) - has to be used in a predefined fashion (not considered in our case, example: information has to be stored in a database, so the interface of the database must be used.

Conformance between Provided and Complete Types.

V. Component Implementation

Description of the inner component structure. Type? - In theory there might be different implementation that conform to the same description as in MDA. But since in most cases, this is a one to one relationship. We don't care.

Basic Component Service effect specification

Composite Component Assembly/Delegation Connector Today: theoretical construct no physical representation.

Conforms Relation to the Complete Type

– Overview –

Benefit: Parametric Contracts (Example)

VI. Deployment

The step of placing or installing software on the target systems. This includes configuration steps if necessary.

General understanding for software components: Placement of a component in an execution environment.

Our understanding: The deployment of a software component defines its context. 1. Assembling of components (Connections) 2. Allocation of components on resources

Both steps can be done independently (two roles, different persons). Nowadays they are generally considered as deployment.

A component can be embedded into different contexts, since it is a unit of independent deployment. This can happen several times within the same system. Leaving type theory.

Within the same system, we have that the same component has different contexts, including the wiring, the mapping to resources and the containment.

Explicit modelling of the context. Two dimensions: Assembly and Allocation, Computed vs. Specified Values.

–Table with different context aspects–

Assembly and allocation can contain special configurations of a component.

Connections and their deployment: Today, either fixed connection between components or naming services. Both do not allow individual connection of components. So, components are no real units of independent deployment at the moment. Technical realisation required.

Composite Components can only be deployed as one piece. Resource diagram and assembly diagram are specified independently.

Communication nodes, which only model network structures.

VII. Deployment of Connections

Deployment of Connections on paths (a path describes the route between two nodes, acyclic)

– Overview –

VIII. Components at Runtime

Cardinalities

Concurrency modelling

Benefit: QoS Prediction

IX. Conclusion

Future Work - MOF-Schema and OCL-Constraints - MDA-Transforms - Description approximating the run-time view of the system (virtual interactors) - Modelling of concurrency, asynchronous method calls - QoS analysis - Process model for CBSE with the Palladio Component Meta Model

Acknowledgments

Thanks to everyone.