

Architectural Considerations in Predictable Component Assembly: A Service-Oriented Perspective

Cobus Smit^{1,2}, John Muller^{1,2}, Jay van Zyl² and Judith Bishop¹

¹ Computer Science, University of Pretoria, Pretoria, 0002
{csmit, jmuller, jbishop}@cs.up.ac.za

² SystemicLogic Innovation Center 222 Grosvenor Road
Bryanston 2194, Johannesburg, South Africa
jay@systemiclogic.com

¹Abstract Software components are deployed in tightly coupled environments within the context of a component model. Techniques such as Prediction-Enabled Component Technology (PECT) [4] combine component technologies with analysis models to address the issue of consumer trust in the quality of assemblies. Service-oriented thinking and challenges surrounding distributed environments are not new, yet they raise certain prominent issues that need to be addressed when crossing organizational and deployment boundaries. This paper describes a common architecture for enterprise application assembly where the focus lies with components and service-assemblies. We consider the aspects surrounding predictability using a service-oriented perspective. Contemporary industry technologies such as .NET and J2EE are used to present a case study of predictable assembly in a broader service-oriented architectural scope. Additional aspects are identified when considering a service-oriented perspective.

1 Introduction

Themes such as electronic business and business process management (BPM) vastly contribute to the increasing effort to connect computer systems [6]. As joint business opportunities present themselves (e.g. Bank-Assurance – Banking and Insurance), enterprises find it necessary to adapt or build applications requiring the use of or integration of new ‘foreign’ software capabilities. COTS (commercial off-the-shelf) related problems and solutions to requirements and acquisitions [7] also come into play. Cross-organizational communication concepts are not new at all. Concepts such as

¹ This work was supported by THRIP Grant No. 2788 from the Department of Trade and Industry.

federated system-level components [2] address information processing needs of multiple end users who may belong to different organizations. Some considerable investment has also been placed into realizing message-oriented middleware and technologies such as CORBA, ultimately connecting these companies. EAI (enterprise architecture integration) solutions prove to be expensive, proprietary and complex in nature [8].

Several independent organizations have their own set of systems ranging from legacy to more modern component-based systems. Successful component technologies, supported by solid business cases [5], are deployed in tightly-coupled environments within the context of a component model and its related business domain. A major challenge in software component technology adoption however, lies with the lack of consumer trust in the quality of assemblies [1]. Foreign software is even less trustworthy when companies consider selecting components. Prediction Enabled Component Technologies (PECT) [1] is an implementation of Predictable Assembly from Certifiable Components (PACC) [4] and combines component technologies with analytic models to address issues relating to consumer trust.

Service-oriented approaches present a different view on component deployment. Component assembly is unlikely outside of its own technical infrastructure [3]. Software-as-services can be viewed as an alternative way of realizing software systems, making the need for software deployment to specific infrastructures unnecessary.

This paper considers predictable assembly from a service-oriented, common architectural perspective. We highlight contemporary enterprise technologies such as J2EE™ (Java™ 2 Enterprise Edition) and Microsoft.NET™ in the context of the presented common architecture.

2 Predictable Assembly

Interfaces and contracts are not well equipped to express extra-functional properties [9]. Several techniques exist to describe extra-functional properties (e.g. manners and analytic models).

2.1 Prediction-enabled Component Technology (PECT)

PECT [4], illustrated in Fig 1, combines a component technology with analytic models. The component technology consists of a component model with one or more runtime environments. The model itself describes allowable component types, interaction mechanisms, runtime environment services and constraints among them. Different runtime environments for the same component technologies enforce the same component model, but may differ in terms of quality attributes [3]. An abstract component technology that models one or more actual component technologies and related tools, and supporting construction activities, constitutes the construction framework. The construction framework is linked to one or more reasoning frameworks by means of an

interpretation that translates the notations between the construction and reasoning frameworks. Reasoning frameworks support prediction by encapsulating property theories and associated reasoning procedures. Finally, the property theory is a proxy for some computational theory.

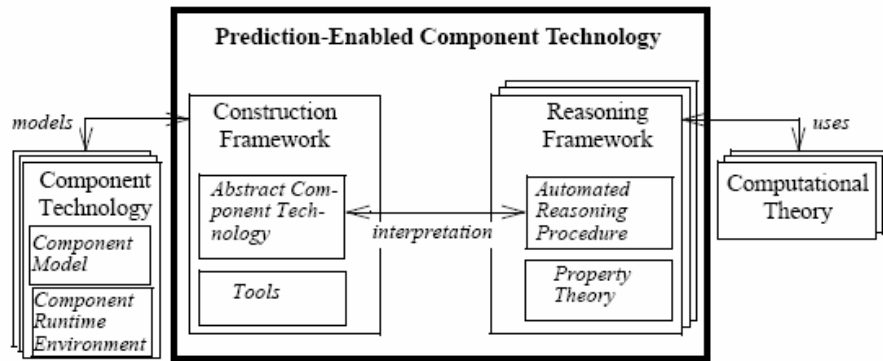


Fig 1. Logical Structure of Prediction-Enabled Component Technology (taken from [4]).

According to [4] the component model imposes constructive constraints on components through the use of APIs (application programmer's interface), concurrency and memory management conventions, etc., as well as analytical constraints, which if satisfied will ultimately result in predictable components. The complementary roles of constructive and analytical constraints are illustrated in Fig 2.

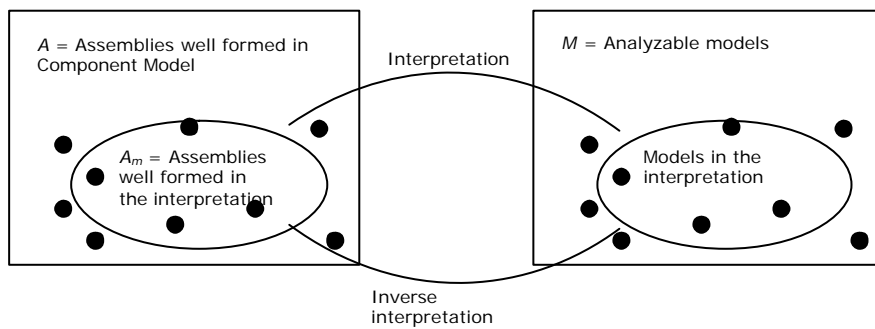


Fig 2. Complementary roles of constructive and analytic constraints (taken from [4]).

Fig 3 informally illustrates the concept of co-refinement. When a component model is expanded, design weaknesses and implementation restrictions are removed, increasing the number of assemblies. Restriction has the opposite effect. The weakening of

analysis models leads to the removal or weakening of assumptions of property theories – the opposite action increases the scope of the property theories to include more assemblies with some associated tradeoffs (loss of precision and reliability of predictions). PECT 1, PECT 2 and PECT 3 simply show three different co-refinements. Co-refinement becomes important when different frameworks are applied to the same assembly.

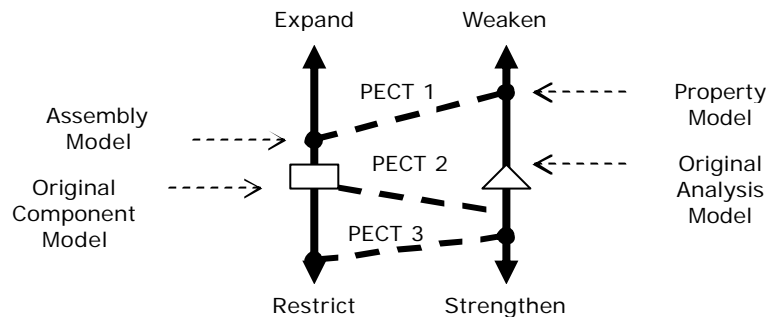


Fig 3. Integration by Co-refinement (taken from [1]).

Weakening analysis models leads to the removal or weakening of assumptions of property theories – the opposite action increases the scope of the property theories to include more assemblies with some associated tradeoffs (loss of precision and reliability of predictions). PECT 1, PECT 2 and PECT 3 simply show three different co-refinements. Additional aspects are introduced when necessary in subsequent sections.

3 Common Architecture and Separation of Concerns

We now define a common architecture for abstraction, supporting the understanding of the implications of assembly and deployment of services in the service-based architectural context. To more accurately describe ‘service-based architectural’, it is necessary to describe the Separation Continuum. The Separation Continuum [12] in Fig 4 combines the concept of vertical and horizontal separations.

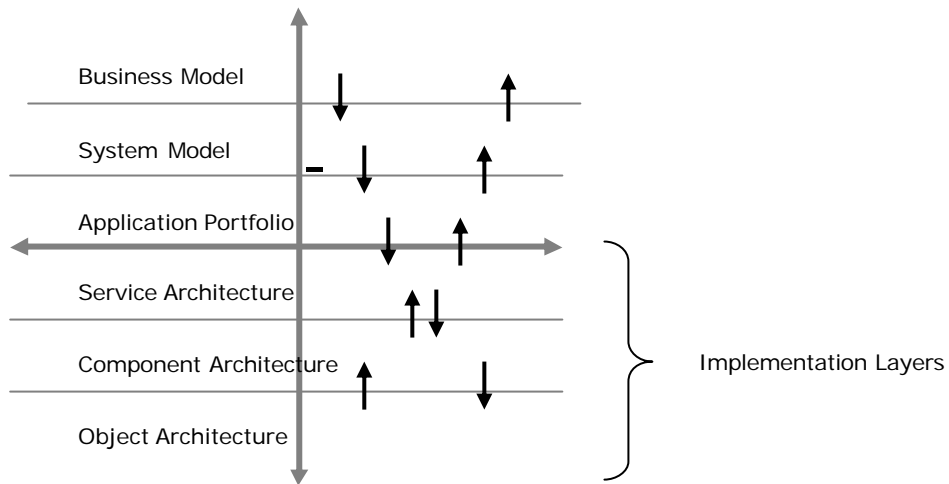


Fig 4. The Separation Continuum and product line realization.

Horizontal separation concerns dependencies relating to user interface and (for example) connection technologies. Vertical separation refers to the layering required to separate platform elements from higher levels of abstraction needed at application levels. Section 3.1 and 3.2 gives an overview of the dimensions involved. Fig 4 describes bottom-up (existing reusable software assets) and top-down (build a system based on the business vision) approaches for constructing application portfolios. We will use the Separation Continuum and continuum as interchangeable concepts.

3.1 Vertical Continuum

The vertical implementation layers related to this work can be defined as follows:

1. Applications Portfolio – Applications are necessary to automate required business systems.
2. Services Architecture – This architecture is concerned with the breadth of functionality needed to implement the business systems via the applications portfolio.
3. Component Architecture – The component architecture view presents a complete view of the business components that can be exposed as services. Components are typically coarse-grained software elements that satisfy a particular requirement.
4. Object Architecture – Components are created from object or other technologies and exposed as business components. Objects are the finest granularity of functionality in a typical system.

3.2 The Horizontal Continuum

The horizontal dimensions (functioning on the implementation layer only) include:

1. User interface: the interface that is presented to the user of the system that might include a multitude of devices including web browsers, personal devices, etc.
2. User interface controller: this controller is needed to manage the behaviour of the interfaces (for example, web pages) that connect to the business logic layer.
3. Services and business logic layer: the ability to have a cluster of business and/or technical services exposed to the user interface.
4. Connection or middleware layer: the ability of the user interface to connect to a server that directs the way in which the interface is used.
5. Data provision layer: the ability to store in a reliable fashion that may be used by the services layer to deal with transactional, system and meta-data definitions.

Different assemblies can now be constructed to fit the purpose of each of the dimensions described. Wiring those services together in an application portfolio is executed on the service level of abstraction.

3.3 Service Level Abstraction

We now bridge the gap between different assemblies by considering a standards-based message-oriented middleware approach. The vertical and horizontal dimensions described form the basis for the discussion surrounding service levels of abstractions. When moving to the service architecture level, the term service needs to be defined. Services and the service concept are not new. CORBA-based standards have a clearly defined means of using software-as-services (e.g. persistence, transactions, security etc.)[11]. Services in our discussion are analogous to yellow pages where service requestors find an appropriate service in a service directory and then connect to that service via a broker. Service providers register their services with a broker. Current day examples such as UDDI (Universal Description, Discovery and Integration) apply. The presence of a consumer, broker and service provider constitutes a service-based architecture.

Services in the continuum can be described as: *modular, accessible, well-described, stateless, implementation-independent, standards-based and message-oriented*. Properties such as ‘standards-based’ (e.g. XML based), ‘implementation independent’ and ‘message-oriented’ (e.g. asynchronous communication) are all complexity reducing aspects supporting the concept of loose coupling. ‘Accessible’ and ‘well-described’ refer more to aspects surrounding the location of services provided by services providers.

These services essentially separate out three major concerns: interface, implementation and connection technologies. Separation is accomplished by providing interfaces that provide all the necessary descriptions that not only describe component services, but also the binding required to communicate with that service. Technology and infrastructure specific proxies can then be generated in native infrastructure code. Con-

temporary technologies such as web services implement this concept with open standards like WSDL² as the interface and SOAP³.

Components have well-defined interfaces within the context of the component model which forms a basis for exposing loosely coupled services into the service layer (versus fine grained class services). All component services are not necessarily appropriate for use in assembling an application portfolio. These services should typically be course grained for purposes of application construction (e.g. applications such as Customer Relationship Management).

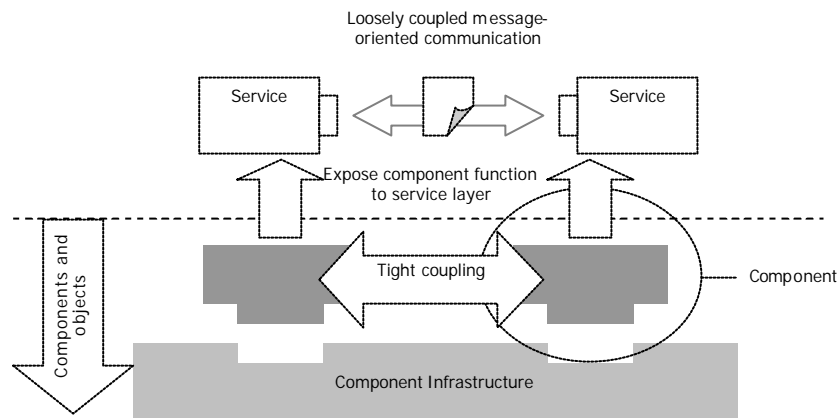


Fig 5. Vertical and Horizontal Abstraction in Component and Service-Dimensions.

Fig 5 illustrates important differences between component-to-component and service-to-service interaction. The components still execute in their native runtime environments, but are now connected by yet another layer of abstraction, using message-oriented, standards-based communication. XML messages pass data around typically invoke functions on other components RPC-style or one-way messaging style.

Now that the services are defined and exposed from components, they need to be assembled to form the application portfolio. The services layer itself does not have an infrastructure defined in the same sense as the component layer – the services layer on its own is almost reduced to interfaces and messages sent between components. It is required to now form orchestrated flow processes to essentially make up the application. Earlier research in this space includes OMG’s Process Engine. These processes can be defined in each horizontal dimension (e.g. business-logic process) of the continuum and also span all dimensions. The process can now be controlled by content (e.g. routing information) imbedded in messages and/or be processed by a proc-

² Refer to the work done by the World Wide Web Consortium (W3C), Web Service Description Group on the WSDL (Web Service Description Language) standard:

(<http://www.w3.org/2002/ws/desc/>)

³ Refer to the work done by the World Wide Web Consortium (W3C), W3C XML Protocol Working Group on the SOAP standard: (<http://www.w3.org/2000/xp/Group/>).

ess manager – contemporary message standards such as BPEL4WS⁴ and WSFL⁵ apply. Given the nature of services and process characteristics it is possible to consider predictable assembly in the service-oriented context using the continuum.

4 Predictable Assembly and Architectural Considerations

With a common architecture defined using a standards-based message-oriented middleware it becomes apparent that predictable assembly now has to be considered in a much broader context. Based on descriptions in section 2 it is now crucial to reassert important aspects of PECT:

1. Analysis models permit analysis and prediction of assembly level properties prior to component composition [4]. In this paper we are interested in addressing behavioural composition using the service level horizontal dimensions of the continuum.
2. Assemblies of components are amenable to one or more methods of predicting emergent (assembly level) properties. The services are exposed from such components and should also be amenable to predicting assembly level properties. Section 4.1 investigates aspects surrounding this.
3. Components are trusted and certified. Trust from a service-oriented perspective is investigated.

Other aspects in our discussion include topics for future research.

4.1 Predictable Service-Assemblies

Services are exposed from components. Services in a service-assembly should also be certified and trusted. It is therefore important to show the implications of exposing components and component assemblies to the services layer. The service-assembly (application portfolio) consists of separate assemblies or single components that may each belong to different environments, each having their own associated reasoning frameworks. To determine predictability over the combined horizontal dimensions, inconsistencies across these different reasoning frameworks have to be resolved. The PECT co-refinement procedure is a useful way to explore the strengthening or weakening and expanding or restricting of analytical model and component models. Horizontal differentiated domains generally utilize similar platforms and products presenting the opportunity to define reasoning frameworks per domain, thereby allowing for a certain level of predictability. Fig 6 simplistically illustrates the assembly of components into constructing the application portfolio. Combinations of reasoning frameworks present a challenge. Different organizations will not necessarily have the same

⁴ Business Process Execution Languages for Web Services is a standard being worked on by Microsoft, IBM, and others

(<http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/>)

⁵ Refer to the work done by IBM on the WSFL (Web Service Flow Language) standard

(<http://www.w3.org/2002/ws/desc/>).

number of reasoning frameworks applied. More applied reasoning frameworks lead to more restricted component models because of all the assumptions that are made about it in order for it to be predictable. It is safe to argue that it is unrealistic to consider all combinations of reasoning frameworks. Fig 6 illustrates the assembly of different predictable component into the application portfolio.

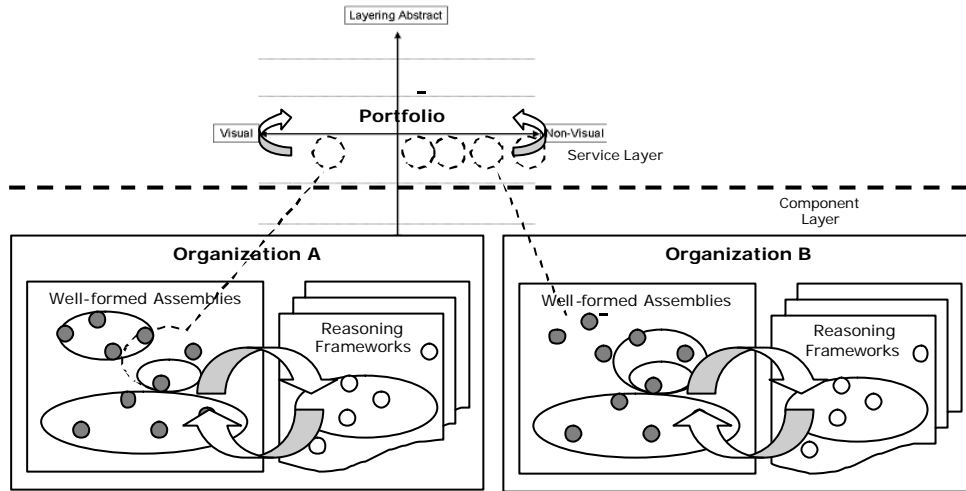


Fig 6. Service Assembly from separate well-formed assemblies.

Property theories such as performance and reliability aspects associated with component assemblies and constituents must be viewed in the context of the service level of abstraction. Reliability in terms of execution guarantees (request semantics [11]) are subject to the vulnerabilities in network protocols. HTTP is considered unreliable as a network protocol. HTTP⁶ addresses aspects in this regard using persistence techniques. Performance will be influenced by network latency (services will be provided from different geographical locations). Predictability will be influenced by whether services are requested each time via brokers per request. More static service-level assemblies will have more predictable qualities than dynamic ones. This will not be the case if service level contracts can be defined to include performance figures i.e. execution time and additional network latency – this will again be influenced by relative broker and flow process-manager locations. Other more cumbersome aspects such as security can be considered to be both functional and non-functional. Security becomes important when access is provided to potentially critical resources across organizational boundaries. Security at the platform level (e.g. encryption and authen-

⁶ Refer to the work done mainly by IBM on HTTPR - Reliable HTTP (HyperText Transfer Protocol) (<http://www-106.ibm.com/developerworks/webservices/library/ws-phht/#h23875>), and keep an eye on WS-Reliable Messaging, investigated by OASIS (Organization for the Advancement of Structured Information Standards) (http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsm)

tication) is non-functional while the application level view is functional (e.g. role-based access).

4.2 Service Reuse and Design Considerations

Will the future exposure of components to the service-layer (for access across different organizational boundaries) influence the initial design of the component? Not all components that are deployed in a component environment are appropriate for exposure. Granularity plays a role here. If we reason in terms of the presented common architecture we will find that component and service technologies build on their finer grained constituents (objects). These fine grained services are more technical or supportive in achieving the portfolio-driven goal in the service layer. Other service level services and assemblies may exist that are technical in nature (e.g. message transformation services using XSLT) – these services form part of the connection dimension. We however consider coarser grained assemblies aimed at satisfying the characteristics of the separate horizontal dimension of the continuum.

The reusability of services in the product line context will be determined by the commonalities and variabilities in the service architecture across organizational borders. Collaborative business cases are also fundamental to success. Reusability can be achieved through collaborative business strategizing and resulting technical design initiatives. Exposing a service to be competitive (commercially) based on analytic criteria will also influence design (e.g. optimization).

It is important to note that we encourage collaborative initiative for purposes of reuse in the functional product-line oriented sense. We do also encourage the use of established domain reasoning frameworks, thereby avoiding the fragmentation of analytic models.

4.3 Third-Party Certification

Predictable service-assemblies will require analytical models. Allowing uncontrolled compilation of these models will again contribute to inconsistencies in reasoning frameworks. Even collaborative initiatives will cluster reasoning frameworks into larger sets, but will not necessarily contribute to building standards. Major industry players typically play a big part in setting up standards in this regards. Third party involvement will enable the development of uniform domain analytical models, without the proprietary deviations introduced by commercial goals. This also means that companies should have competency to deal with aspects of certification.

5. Case Study

J2EE™ and .NET™ both provide built-in capabilities for loosely coupled communication. The integration of web service capability has been (and still is) pursued rigor-

ously by industry. Our earlier work has shown the technologies can be connected [10]. Both technologies incorporate open standards to enable service-level realizations based on component platforms. Observations made in financial industry implementations using J2EE™ and .NET™ now lead us to show the separation of technologies in the continuum with emphasis placed on component and service-architectural levels. We now consider these contemporary technologies in the context of the continuum and show issues around previously discussed topics. We also show service-level technological capabilities and considerations in predictability for both J2EE™ and .NET™.

5.1 Architectural Aspects in .NET™

The .NET™ framework provides numerous different technologies that can be classified using the horizontal dimensions of the continuum. Fig 7 shows the major .NET™ technologies each functioning in a dimension matching the characteristics to the dimensions of the horizontal continuum.

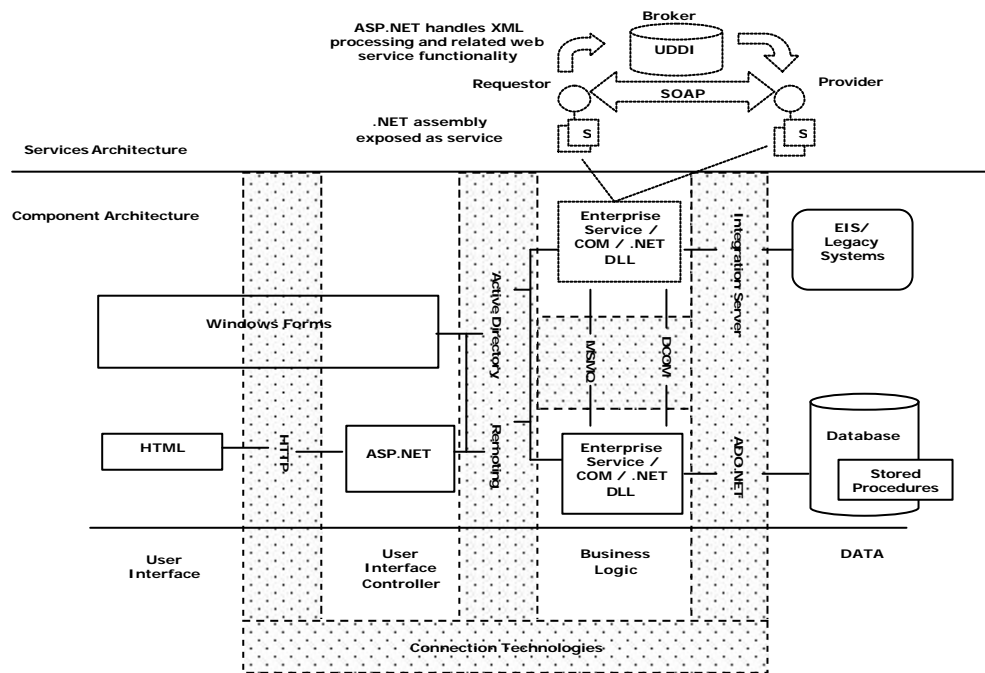


Fig 7. .NET™ in the Separation Continuum.

5.1.1 .NET Technologies in the Continuum

Only components in the business logic section are exposed. .NET™ natively supports the user interface, user interface controller and data layer on the component level. This does not mean that .NET™ is incapable of exposing other dimensions as services – some properties inherent in the provided technologies are just more implicit. Open standard technical committees and drafted standard proposals have only recently been defined in the space of user interfaces – example such as WSXL⁷ and the more recent WSRP⁸ apply. Native programmatic framework support is not yet available to realize these newer standards.

ASP.NET provides component level user-interface-controller capabilities to enable message processing (e.g. SOAP) between services. Services are exposed by deploying certain framework specific configuration files and service components. Orchestrated process construction is either executed manually by building components that process messages with routing logic or by using commercial products such as Microsoft® BizTalk™. Experimental software such as Web Service Extensions integrates open standards such as WS-Routing, WS-Security and WS-Attachments.

Connection layers technologies such as Microsoft® MQ™ that are also able to present service-level interfaces enabling users to place messages on a queue.

The service layer of abstraction is independent of the component model as illustrated in Fig 7 and Fig 8. It is however the component model capabilities that enable the service-level abstraction.

5.1.2 Levels of Predictability in .NET™

Many companies are using alternatives to J2EE™ and are switching over to or are at least partially integrating .NET™ software capability (e.g. front-end development). The .NET™ framework itself has integrated web-service support provided by ASP.NET. Integrated support for certifiable services and service assemblies will in future enable service assemblies to be more predictable. .NET™'s own integrated support for web services provides higher levels of confidence as there is no reliance (and so – external dependencies) on third-party vendors to provide service level capabilities. It is the integrated support for service-level capabilities that also drives the industry leader to renew integrated support.

⁷ IBM forms the main contributor for the WSXL (Web Service Experience Language) standard (<http://www-106.ibm.com/developerworks/library/ws-wsxl/>).

⁸ OASIS has released the WSRP (Web Service Remote Portlet Specification) (http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrp)

5.2 Architectural Aspects in J2EE

J2EE™ is similar in the way that it maps to the horizontal continuum, as illustrated in Fig 8. J2EE™ is different in its implementation approach to .NET™.

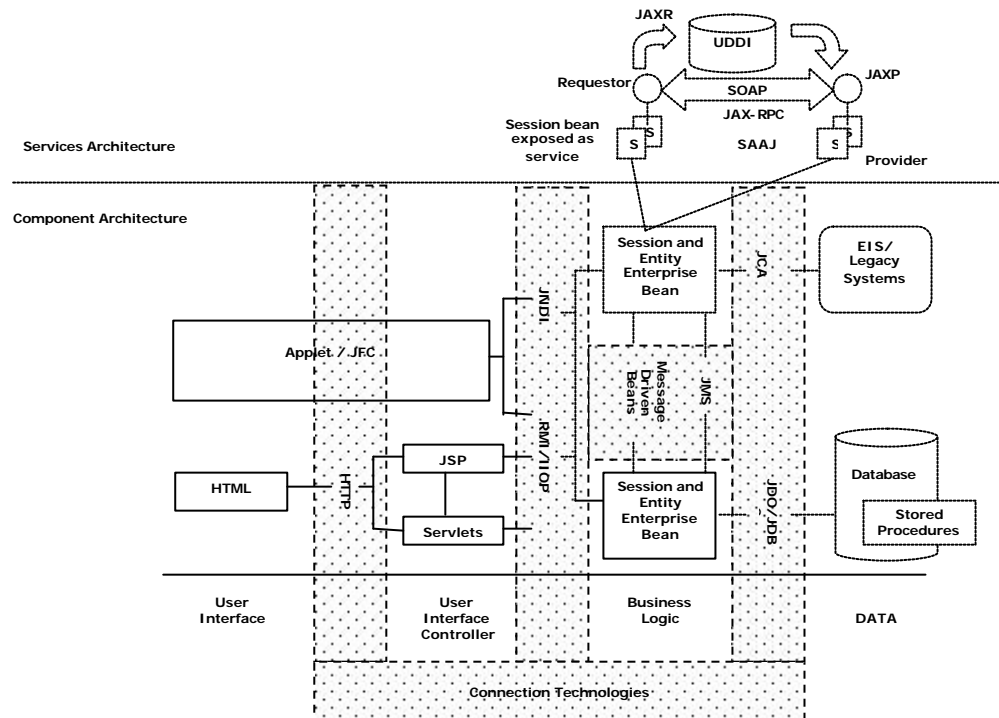


Fig 8. J2EE™ in the Separation Continuum.

5.2.1 Technologies in the Continuum

Fig 8 depicts J2EE™ in the continuum on the component and service architectural layers of abstraction. Not all technologies are exposed as services. Many standards are still being defined in this space as described in section 5.1 (e.g. user interface services). As with .NET™ user interface controller technologies such as Servlets act as message processors using the HTTP protocol. We only show the exposure of Session Bean EJB technologies as it is one of the most used service-level capability enablers. Connection technologies are also exposable (JMS service interfaces). Database stored procedures are also exposable as services or directly accessible via entity components (e.g. Entity EJB).

Assembly and orchestrated process control is similar to .NET™ in that message content can be processed manually using logic embedded inside components (e.g. Servlets). Commercial tools such as IBM® WebSphere® - Integration Edition enable

service orchestration capabilities. IBM® Emerging technologies initiatives also present open standard support.

5.2.2 Levels of Predictability in J2EE™

J2EE is a defined standard that is implemented by several different vendors (e.g. IBM, Oracle and BEA to name a few). To gain competitive advantage, tools and additional proprietary libraries are added to simplify or enable new software capabilities. The component paradigm has been around for quite some time – and so has J2EE™. Component level development in J2EE™ is considered to be quite mature when technologies like EJB are considered. Service-level capabilities have only been a recent addition to J2EE™. XML processing APIs and supporting community process technologies were never part of the core J2EE distributions, leading vendors to implement proprietary solutions. Newer developments show that J2EE™ has integrated more APIs and web service capability enablers into its core platform. Competition between different vendors (leading to varying interpretations of reasoning frameworks) and still existent external dependencies will definitely influence predictability in service assembly construction. Based on these factors J2EE™ will provide lower levels of confidence in predictable assembly despite its commercial success and global popularity.

6 Future Work

This paper covers a wide range of aspects ranging from architecture, middleware, to predictability in assemblies with the focus on collaborating business enterprises. Each of these topics is large and worthy for future research in their own right.

It is our future focus however to continuously investigate the correlation between component and service-related behaviour in order to establish defining characteristics influencing certification and predictability in complex architectures.

7 Acknowledgements

The authors hereby gratefully acknowledge the contributions made by members of the Polelo Research Lab.

References

1. Hissam, A.S., Moreno, G.A., Stanford, J.A., Wallnau, K.C.: Packaging Predictable Assembly. In: Bishop, J. (ed.): Component Deployment (Proceedings of the IFIP/ACM Working Conference, CD 2002). Springer-Verlag, Berlin Heidelberg New York (2002) 108-124.

2. Herzum, P., Sims., O.: Business Component Factory – A Comprehensive Overview of Component-Based Development for the Enterprise. Wiley Computer Publishing, John Wiley & Sons, Inc. (2000)
3. Sims, O.: Service Oriented Architecture: Part 1. The Foundation. CBDi Journal (March 2003).
4. Wallnau, K.C.: Volume III: A Technology for Predictable Assembly from Certifiable Components (PACC). Technical Report CMY/SEI-2003TR-009 ESC-TR-2003-009 (<http://www.sei.cmu.edu/publications/documents/03.reports/03tr009.html>).
5. Williams, J.: The Business Case for Components. In: Heineman, G.T., Councill, W.T. (eds.): Component-Based Software Engineering: Putting the Pieces Together. Addison Wesley Publishers (2001) 79-89.
6. Fremantle, P., Weerawarana, S., Khalaf, R.: Enterprise Services: Examining the Emerging Field of Web Services and how it is Integrated into Existing Enterprise Infrastructures. Communications of the ACM, Vol 45, No10 (October 2002), 77-82.
7. Maiden, N.A.M., Ncube, C., Moore, A.: Lessons Learned during Requirements Acquisition for COTS systems. Communications of the ACM, Vol. 40, No. 12 (December 1997), 21-25.
8. Stal, M.: Web Services: Beyond Component-Based Computing. Communications of the ACM, Vol 45, No 10 (October 2002), 71-76.
9. Crnkovic, I, Hnich, B., Jonsson, T., Kiziltan, Z.: Specification, Implementation, and Deployment of Components. Communications of the ACM, Vol 45, No. 10 (October 2002), (35-40).
10. Bishop, J., Horspool, R.N., Worrall, B.: Experience in integrating Java with C# and .NET. To appear in: Concurrency and Computation: Practice and Experience (2004). Wiley Computer Publishing, John Wiley & Sons, Inc.
11. Emmerich, W.: Engineering Distributed Objects. Wiley Computer Publishing, John Wiley & Sons, Inc. (2000).
12. Van Zyl, J. : A Perspective on Service Based Architecture: The Evolutionary Concept that Assists Technology Providers in Dealing with a Changing Environment. Proceedings of SAICSIT 2002, ACM International Conference Proceedings Series (2002), 249