American International University-Bangladesh (AIUB)

CompaaS

High Performance, Polyglot Component-as-a-Service

Tanjim Hossain (18-38377-2)

Rafid, Kafil Wara (18-38175-2)

Bhuiyan, Asif Iqbal (19-41674-3)

Hasan, Zahid (18-38275-2)

A Thesis submitted for the degree of Bachelor of Science (BSc)

in Computer Science and Engineering (CSE) at

American International University Bangladesh in May, 2023

Faculty of Science and Technology (FST)

Abstract

The purpose of this thesis was to build a theoretical basis of a platform - that can manage the whole life-cycle of polyglot software components, and their interconnectivity in a multi-cloud environment, in a highly scalable, secure, and efficient way.

To approach our goal, we studied the Component Based Software Engineering (CBSE) literature, evaluated the existing component models, programming paradigms, platforms to get inspirations and valuable insights - by reading academic papers, authoritative books, talks from original authors, and experimentations.

Our primary motivation for doing this was to reduce the architectural in-efficiency and complexity of the mainstream FaaS approach, by unifying the CBSE and Actor Model. This unification ultimately results in more code reusability, better observability and reasoning, and less cloud-cost. The key result from our extensive research validates that to a great extent.

One of the key factors to the efficiency of our solution is utilizing Language Virtualization - that enables us to co-locate components written in different languages in a single JVM instance, which significantly reduces Inter-Component Communication cost. Following Thread-Per-Core, dynamic scheduling strategies, and compartmentalization - we can achieve stable and efficient utilization of compute resources, and balance throughput / latency maximally. Also, a key outcome from this thesis is an opinionated Component Model, that embraces the complex, evolving dynamicity of software systems via Extreme Late-Bindings.

The insights from this thesis gives a practical basis for a more robust, scalable path towards CBSE - that solves some major issues with mainstream FaaS, Microservice like approaches (e.g., latency, redundancy, complexity). Our proposed solution makes CBSE more practical, and approachable with ease in this cloud native era. Component reuse should significantly reduce overall software development cost, along with required human effort. The design of our component runtime should also reduce energy consumption of cloud infrastructures.

Declaration by author

Approval

The thesis titled **“CompaaS - High Performance, Polyglot Component-as-a-Service”** has been submitted to the following respected members of the board of examiners of the department of computer science in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science on **(03/05/2023)** and has been accepted as satisfactory.

Contributions by authors to the thesis

This thesis is a result of time and effort from 4 authors. Below table gives a concise view over the percentage of their contribution:

[redacted]	[redacted]	[redacted]	[redacted]	
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]

Acknowledgments

We would like to express our sincere gratitude to our supervisor Kishor Morol - for his patience, support and also for the flexibility that he gave us throughout the development of this thesis. We would also like to thank Shafqat Ullah from Ālo Labs [42] - for his valuable feedback and insights.

We extend our appreciation to all the prior works done in this field and their authors. Also, the revolutionaries like Alan-Kay, Carl-Hewitt, Gul-Agha, John-McCarthy for being the inspiration of our work.

Finally, we thank our families for their love, support, and encouragement, which were the foundation of our academic pursuits.

Thank you all for your contributions to the success of this thesis.

Keywords

CBSE, Actor Model, Component Model, Language Virtualization, GraalVM, Akka, Component, Modularity, Distributed System

Table of Contents

List of Abbreviations and Symbols

Abbreviations	
CS	Computer Science
CSE	Computer Science and Engineering
CBSE	Component Based Software Engineering
TPC	Thread Per Core
FaaS	Function as a Service
CompaaS	Component as a Service
IPC	Inter Process Communication
CQRS	Command Query Responsibility Segregation
PoC	Proof of Concept

Chapter 1

Introduction

1.1 Overview

The software industry is experiencing significant growth, with new software systems and applications being developed at a rapid pace. New Programming Languages, Libraries and Frameworks are being introduced every other day - that involves porting the existing tools to the new environment, so they can work seamlessly with the new language or framework. This results in a massive waste of human effort, and software clutter.

Component Based Software Engineering (CBSE)[43] has the potential to resolve this problem. CBSE is a discipline that advocates for componentizing software modules in a modular way. This modularization can significantly increase code re-use, saves time and resources, reduce complexity, increase overall quality of large systems, and lead to faster development.

But, without commonly agreed upon standardization CBSE fails to meet its potential. Too many competing Component Models from different vendors makes Component approach difficult to adopt. Also, many of the Component Models are designed around a specific language / runtime / platform. That implies a component built conforming to any specific component model needs extra effort to be ported to another model, or even requires complete rewrite.

Component Model agnostic platform or runtime would fix the modularity aspect of the problem, but faces a new problem - inefficiency. A Component written in one language needs its own runtime, and requires some form of IPC to communicate with another Component written in a different language. This is a major drawback, that results in IPC latency along with extra processing overhead of serialization and deserialization of the communicated message/data. As a system grows more complex, involving more components - this overhead accumulates and renders unacceptable performance. FaaS and Microservice like systems face this issue a lot[44], as to process a single end user request, function/services has to get results from others, causing a chain of calls. Each of those calls has to pass through multiple layers of abstractions, causing the overall processing time for the request higher than the user acceptance.

Our aim is to resolve this aspect of CBSE. The problem is twofold: first, there are too many Component Models, but very few are generic enough for being usable in programming in the large and scalable way. second, a lack of efficient distributed runtime / platform that can schedule the components and establish the interconnectivity.

We solve both of them - by proposing a new Component Model that unifies some major mainstream and old paradigms / ideas like Actor Model[45][46], CQRS, FSM / Statechart, Event Sourcing etc. Also, our approach utilizes Language Virtualization with Partial Evaluation via GraalVM. This unification and combination is powerful and gives some unique value propositions that do not exist currently in practice or academically.

Some of the key research questions that is answered through the findings of our research are:

1. What are the major obstacles towards CBSE adaptation in large scale systems?
2. How to model Components for maximal reuse?
3. How to schedule Components in multi-cloud infrastructure for optimal performance?
4. What are the implications of modeling Components as Services?

The forthcoming chapters will go into the details of the problems, solutions and their evaluation.

1.2 Objective

In this thesis we are aiming to design and develop a theoretical basis for a platform that acts as a managed host for Components - that conforms to our opinionated Component Model.
More precisely, In this thesis project, we want to achieve the following goals:

- Have a thorough understanding of the CBSE landscape, and case studies of FaaS, Microservice usage in the industry.
- Extract the best out of existing Component Models, and design our own if necessary.
- Figure out how to schedule the Components in a multi-cloud environment.
- Evaluate the options of tools to choose from - as building blocks for the platform itself.
- Evaluate our solution in contrast to the existing alternatives.

Chapter 2

Literature review

The development and delivery of software has changed as a result of the rise of cloud computing. Component-as-a-Service (CompaaS), one of the numerous techniques to develop cloud-based software, has drawn attention as a result of its promise to improve the modularity, reusability, and scalability of software systems. This literature review explores the most recent, cutting-edge research on CompaaS with an emphasis on its key ideas, advantages, difficulties, and practical applications. This study attempts to provide a clear understanding of CompaaS and its possible impact on software development processes through a thorough assessment of the pertinent literature.

2.1 Component Based Software Engineering

A method of developing software called component-based software engineering (CBSE) entails putting together reusable, stand-alone software components to create software systems. The component model, component classification, and component composition are the three main components of CBSE. A standardized framework that describes the structure, function, and interaction of software components is offered by the component model[1]. Component classification categorizes components based on functionality, granularity, and deployment. Functional classification groups components based on the specific tasks they perform, while granularity classification distinguishes between fine-grained and coarse-grained components. Deployment classification differentiates between local and remote components.

Component composition involves combining components through techniques such as inheritance, aggregation, and assembly. Inheritance-based composition allows for extending existing components, while aggregation-based composition combines components to achieve specific goals. Assembly-based composition enables dynamic configuration at runtime [2].

CBSE offers benefits such as increased productivity and improved software quality.

2.2 Object Orientation - Alan Kay

Alan Kay, a prominent computer scientist, made significant contributions to object orientation. His ideas focused on message passing, extreme late-binding, and the analogy of objects as individual computers.

Message Passing:
Kay emphasized message passing as a fundamental mechanism in object-oriented systems. Objects communicate by sending and receiving messages, enabling modularity and encapsulation. Smalltalk, a pioneering object-oriented programming language developed by Kay, heavily incorporated the message passing paradigm [3].

Extreme Late-Binding:
Kay introduced the concept of extreme late-binding, allowing dynamic determination of method behavior at runtime. This flexibility enables objects to adapt and respond to changes dynamically, enhancing the extensibility and adaptability of object-oriented systems [4].

Object as Individual Computers Analogy:
Kay analogized objects to individual computers, envisioning them as self-contained computational units. Objects can communicate by sending messages, similar to computers in a network. This analogy highlights encapsulation and the autonomy of objects while enabling collaboration.

2.3 Functional Programming

Functional programming (FP) is an approach to develop software which uses pure functions for creating maintainable software. In other words Functional programming is its programming paradigm where a program can be created by using applying and composing functions. In the Functional programming (FP) paradigm everything is bound using only pure mathematical functions based on Lambda calculus. Haskell programming language used in functional programming. [5][6]

Monads:
A monad is a structure which combines program functions or fragments In functional programming. In other words it is a common concept that helps with operations between pure functions with the side effects. Mathematician and logician Haskell Curry used the term “monad” to describe it as a way to structure functional programs.In FP Monads provide a convenient framework to simulate effects such as output, or non-determinism, exception handling,global state.

A monad contains 3 building blocks

1. For producing computation a constructor .
2. A function for taking values .
3. Another function to take 2 computations to perform them one after another.

Haskell provides some pre-built definitions in their core libraries for some general monad structure and also for common instances. [7][8]

2.4 Actor Model

The actor model is a mathematical conceptual simultaneous computation model which treats an actor as the basis for constructing a block of concurrent computation In computer science. The Actor Model also defines some common rules like how to interact with each other and how the system's components should behave . It can make local decisions like create more actors, send more messages, and determine how to respond to the next message it has received.

Erlang:
Erlang is a general-purpose, concurrent, functional high-level programming language or we can say functional programming language and runtime environment system for building massively scalable software in real-time systems with requirements on high availability. It was designed to write concurrent programs that can "run forever" with concurrent processes to structure the program.

Akka:
Akka is a runtime environment and also a toolkit for building highly concurrent, distributed applications and softwares on the Java virtual machine .Even though it works on JVM, Akka is written on Scala and the binding is provided for both Java and Scala. Akka's Approach for handling concurrent applications is based on The Actor Model.

Actor System - Gul Agha:
A hierarchical collection of actors known as an actor system share common configurations such remote capabilities, deployments, dispatchers, and addresses.The Actor System is the foundational actor in the actor model.

2.5 PITL and PITS

Programming In The Small:

Programming in the small is a term that describes the activity of writing a small program that performs one or a few tasks very well or we can say it describes the activity of writing a small program. It is usually done by a single engineer or a small team, without following a formal software development methodology. Programming in the small can benefit from applying some simple guidelines and best practices to improve the quality and maintainability of the code. Small programs are typified by being small in terms of how quick to code and typically perform one task or a few very closely related tasks very well; their source code size is easy to specify.[9][10]

An examples of small programs could be -
A program that creates pyramid patterns using loops
or, A program that swaps two variables without using a third variable.

some of the benefits of writing small programs are:

- They are easier to write, test and debug than large programs.
- They can be reused in other programs as modules or subprograms.
- They can perform one or a few tasks very well and efficiently.
- They can be understood by other programmers more easily.
- They can save time and resources by avoiding unnecessary complexity and overhead.[11]

Programming In The Large:

Programming in the large is a term that describes the activity of writing a large program that consists of many small programs (modules) that interact with each other. It is usually done by a large team of engineers, following a formal software development methodology. Programming in the large can benefit from applying some techniques and tools to manage the complexity and coordination of the system.[11][12]

Some of the techniques and tools for programming in the large are:
- Modular programming: a technique that decomposes a large program into smaller units (modules) that have well-defined interfaces and dependencies[12].
- Dataflow programming: a technique that models a program as a network of independent components that communicate through data streams[2].
- Version control and configuration management: tools that help track and manage changes to the source code and its dependencies[23].
- Testing and debugging techniques: tools and methods that help verify the correctness and reliability

of the program and identify and fix errors[23].
- Profiling and code improvement techniques: tools and methods that help measure and optimize the performance and resource usage of the program[3].
- Automatic parallelization compilers: tools that help transform sequential code into parallel code that can run on multiple processors or cores.

The design of the overall structure of a program is what is called “programming in the large” while “programming in the small”, which is sometimes called coding, would then refer to filling in the details of that design [13]

2.6 Microservice and Serverless

Microservices are designed to split up large, complicated monolithic programs into smaller, simplified services. Microservices are designed to make the process of using them to create complicated applications simpler [14]. Each microservices can be either stateless or stateful. To function, a stateful microservice must carry some sort of its own state. Next actions are processed in contrast to previous transaction memory, and these "memories" are used to carry out the next transactions whereas a stateless function does not hold these “memories” after the completion of a successful transaction. However, both Serverless and Microservices emphasize the system's decoupling, so they can coexist.

Serverless computing is referred to as a "function as a service (FaaS)" or "function-driven event" as FaaS is the primary form of serverless representation [15]. The resources on the platform are launched when a predetermined event is triggered, and it is based on the code written by the developer for precise resource allocation. The most obvious benefit of not having a server is that it frees up resources to be used for application development, infrastructure services, and server maintenance [16]. In an AWS serverless environment, every Lambda function is completely isolated. If one of the features is disabled, it has no effect on the other features and doesn't shut down the server. Datasets are divided by the Lambda architecture to accommodate different types of calculation scripts [17]. FaaS and PaaS platforms often use lightweight VM technology like Firecracker [40], that we can also utilize for our platform deployment and exo-process.

2.7 Language Virtualization

A method for separating a service from the underlying physical delivery of that service is virtualization. A programming language can only be virtualized with respect to a class of embedded languages if it can give these underlying languages an environment that renders integrated executions, with little more work than creating the simplest complete integration, nearly comparable to equivalent stand-alone language implementations in terms of expressiveness, efficiency, and safety[36]. The usage and flexibility of hardware can be increased by running different operating systems and programs concurrently on the same computer and its hardware with the aid of virtualization. A single physical instance of a resource or application can be shared by several users at once through virtualization. It enables this by giving physical storage a logical name and instantly supplying a pointer to that physical resource.

When only parts of a program's input data are accessible, a technique known as partial evaluation is used to only partially run the program[37]. A partial evaluator removes the first and biggest order of magnitude: the interpretation overhead when given a language specification in the form of functional semantics. One advantage of the strategy is that it creates target programs that are always interpreter-correct[38].

GraalVM is perfect for cloud-native deployment since it compiles in advance, the native version starts up quickly, and it needs a smaller amount of memory. By enabling programmers to access the best resources required to address firm issues regardless of the language they are written in, it promotes multi language projects and boosts productivity. It enhances peak throughput by cutting down on garbage collection time, and reduces program delay[39].

2.8 Event Sourcing

Event sourcing is a data modeling and persistence technique that captures and stores domain-specific events representing changes to an application's state over time. It involves two key components: the event journal and projection.

Event Journal:
All domain events are recorded in the order they happen in the event store, also known as the event journal. It serves as the only source of truth for the application's current state.. Because events are append-only and immutable, replaying them allows for the reconstruction of the application's state[18].

Projection:
Projections process the stored events to derive and maintain the current application state. By consuming events from the event journal, projections update their internal representation of the state. Projections can be optimized for specific read/query requirements, ensuring efficient access to

relevant data. Event sourcing separates read and write concerns, enabling scalability and flexibility in handling complex domain models [19].

Event sourcing has gained attention in various domains, and notable sources include "Event Sourcing and CQRS: A Journey" by Greg Young [20] and "Event Sourcing: Foundations and Trends in Databases" by P. Zdun and S. Schulte [21].

2.9 Event Processing

Event processing is a fundamental concept in the field of data management and analysis that involves the continuous handling and analysis of events generated by various sources. Two important aspects of event processing are the processing pipeline and complex event processing (CEP).

Processing Pipeline:
The event processing pipeline refers to the sequence of stages involved in processing events. It typically includes event capture, filtering, transformation, enrichment, correlation, and aggregation. Each stage performs specific operations to extract meaningful insights from the incoming stream of events. The processing pipeline allows for real-time or near-real-time analysis, enabling timely decision-making and response to dynamic situations [22].

Complex Event Processing (CEP):
CEP is a specialized technique in event processing that focuses on identifying and analyzing complex patterns and relationships among events in real-time or historical data. CEP engines employ rule-based systems or pattern matching algorithms to detect and correlate events that meet specified criteria or patterns. This enables the identification of high-level, composite events that have significance beyond individual events. CEP finds applications in areas such as fraud detection, algorithmic trading, and IoT analytics [23].

Several research papers contribute to the understanding and advancements in event processing. Notable sources include "Event Processing: Designing IT Systems for Agile Companies" by Opher Etzion and Peter Niblett [24], "Complex Event Processing: A Survey" by L. Chen and C. Hsu [25], and "Stream Processing and Analytics" by D. Abadi et al. [26].

2.10 Thread Per Core (TPC) Architecture

The number of threads for developing an application depends on the application requirements that are being built followed by its architecture but in general, applications are composed of one or more threads [27],[28]. To improve traditional practices, thread-per-core architecture emerged to reduce the latency and improve throughput [29].

Resource Management is of 3 types for an application built on the following thread-per-core architecture which is 1. Shared everything approach, 2. Shared nothing approach, and 3. Shared something approach.

Shared Everything: the threads are connected to the CPU cores when the data is preserved in the DRAM which is shared among all the threads in the shared everything approach. The advantage of this approach is that it can max out system throughput as any CPU core can be utilized to process the requests. The problem is that data rebounds between caches.

Shared Nothing: Shared Nothing lets each thread use the resources assigned to the thread. This thread will be responsible for responding to the requests related to the data shared with it. This approach improves CPU cache efficiency and removes thread synchronization.
Shared something: In this approach, the application needs to partition some resources, and more than one CPU core can access the same data.

2.11 Cloud-Native Infrastructure - CNCF Landscape

The infrastructure is made up of all the hardware and software that support apps. This includes all necessary infrastructure and software, such as data centers, operating systems, deployment pipelines, configuration management, etc., to support the life cycle of an application[33]. To execute cloud-native apps efficiently, a cloud-native infrastructure is needed. Even the best cloud-native application can be rendered useless without the proper design and management methods.

The Cloud Native Computing Foundation (CNCF) is assisting in standardizing and opening up the cloud-native industry. Their interactive Cloud Native Landscape map gives us a list of the programs and services that help to make cloud-native computing possible.

Database:
In order for traditional databases to function properly, all the files and resources needed to be located on the same server. Databases for cloud-native apps require data to be distributed over numerous servers because they cannot have a single point of failure. The development of more resilient and adaptable cloud applications is made possible by distributed databases[34]. Data is saved by replication and duplication across various independent servers.

Messaging and Streaming:
The cornerstone of both DevOps and cloud-native computing is the microservices architecture which allows interaction between services while protecting data consistency and averting data corruption.

The middleware of Cloud-native consists of message brokers and streaming services. The fundamental operating premise of messaging is the same regardless of the service or protocol: message or event "producers" submit information to a middleman (the broker), who then distributes it to "consumers" or event receivers so they may act on the information[34]. There is some disagreement about what exactly qualifies as streaming, but to put it briefly, streaming is just the capacity for large-scale communications.

Scheduling and Orchestration:
One of the key operational tasks needed to develop a durable, loosely coupled, and conveniently scalable cloud-native application is the deployment and management of a cluster of containers. Scheduling and orchestration solutions have the advantage of abstracting away the difficulties involved in managing a large number of containers. They provide a framework to deploy, roll back, load balance, and set up self-healing features at scale as opposed to controlling applications at the container level.[34]

Logging, Tracing, Monitoring:
It's crucial to monitor and examine every part of an application to ensure that there are no service interruptions and that any anomalies are quickly identified and fixed. This group of tools is divided into logging, monitoring, and tracing tools.

Logging: To keep track of error reports and associated information, logging programs gather, store, and analyze these messages. Applications continuously send out log messages that describe their current state. These log messages record a variety of system events, including failed or successful activities, audit data, and health events[35].

Monitoring: Creating a contemporary platform requires the collection, storage, and analysis of logs. Logging facilitates carrying out any or all of those responsibilities. When an event occurs, monitoring enables operators to act swiftly and sometimes automatically. It offers information on a system's present state of health and keeps an eye out for modifications[35].

Tracing: Tracing addresses this issue by giving each message delivered by the program a special identification number. It is possible to track or monitor individual transactions as they pass through a system using that special identification. Both the health of the application and the debugging of problematic microservices or activities may be determined using this information[35].

2.12 Inconsistency Robustness

Inconsistency In addition to a departure from the prevailing paradigms of inconsistency denial and inconsistency eradication, which attempted to sweep problems under the rug, robustness is the ability of information systems to execute in the face of persistent discrepancies. Although it has been a progressive development within the Inconsistency Robustness paradigm, deriving contradictions has not been a "game-stopper." It is founded on the idea that contradictions must be made clear in order for arguments in favor of and against ideas to be formalized.[30]

Numerous disciplines, including logic programming, software engineering, and artificial intelligence, can benefit from consistency robustness. For pervasively inconsistent theories of practice, for instance, Inconsistency Robust Direct Logic can be employed.[31]

We need to improve the efficiency of huge information systems that must function in a setting of pervasive inconsistency, hence inconsistency robustness is a desired attribute. By evaluating the costs and advantages of the legislation to all stakeholders, inconsistency robustness can assist in the creation of an improved patent law. [32]

Chapter 3

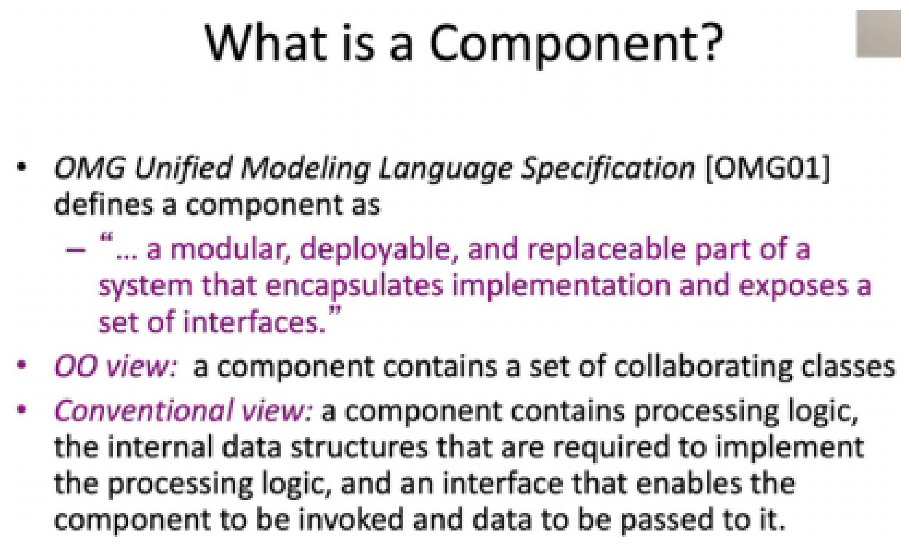
Methods

3.1 Overview

This research was started by observing the industry, gathering insights from leaders in the tech industry, and reflecting on our own experiences. Then case studies were looked into and observations were gathered. For a more detailed rigorous explanation, a lot of relevant academic papers/journals were reviewed. With further analysis and connecting the dots, we built some strong intuition to design the subject matter of this project. Lastly, to have some decent level of confidence in our work, we needed validation. This chapter gives a complete overview of all these steps and how those affected the end result as well.

3.2 Case Study

To begin with, we explored current trends in the computing industry such as AI landscape, Platform Engineering, Serverless and other interesting topics. We found that there was one fundamental common building block involved in all of them - that is Component. This term is quite generic and doesn't have any standard definition. However, as we studied Component Based Software Engineering (CBSE) discipline, we concluded our own definition of the word in our context. It was clear that without proper Component-based approach, software scalability can't be ensured and is bound to obsolescence.


What is a Component?

- *OMG Unified Modeling Language Specification* [OMG01] defines a component as
  - "... a modular, deployable, and replaceable part of a system that encapsulates implementation and exposes a set of interfaces."
- *OO view:* a component contains a set of collaborating classes
- *Conventional view:* a component contains processing logic, the internal data structures that are required to implement the processing logic, and an interface that enables the component to be invoked and data to be passed to it.

To call any part of software (e.g., some code, artifact) as Component, it must conform to some Component Model. We studied the existing and emerging Component Models and analyzed their pros and cons. In our opinion, a good component model must be algebraically compossible, can form dynamic topology, can act as a unit of concurrency, and hide any implementation detail internal to that component.

We were inspired by Actor Model to build such a Component Model that we also used for designing the platform for orchestrating those components.

3.3 Analysis

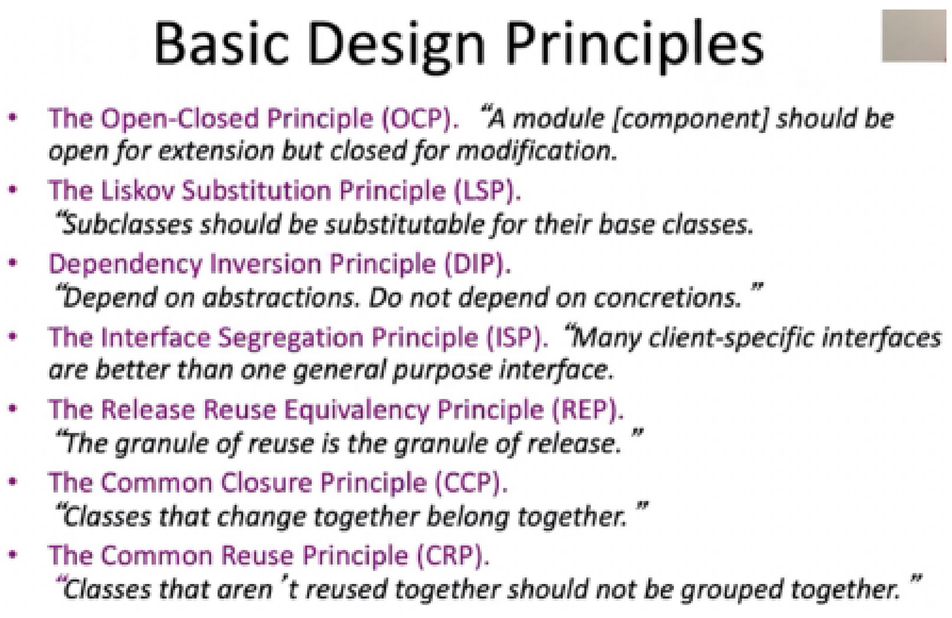
For our goal, we needed to find end-to-end all the components/methods/principles to follow - that we can glue together for the end unique artifact. For that, we had to compare multiple options as alternatives for doing each individual thing. That comparison involved understanding the underlying concepts and approaches - recursively.

3.4 Design

For designing the end result, we tried our best to use First-Principle Thinking - to dig into the essence of each approach, and what value they propose. We had a few goals set, so that any decision must reflect those, that includes:

- Linear scalability
- Abstract away low level details
- Low overhead / cost for abstractions

- Enabling maximum local reasoning.
- Modularity



Basic Design Principles

- The Open-Closed Principle (OCP). *"A module [component] should be open for extension but closed for modification.*
- The Liskov Substitution Principle (LSP). *"Subclasses should be substitutable for their base classes.*
- Dependency Inversion Principle (DIP). *"Depend on abstractions. Do not depend on concretions. "*
- The Interface Segregation Principle (ISP). *"Many client-specific interfaces are better than one general purpose interface.*
- The Release Reuse Equivalency Principle (REP). *"The granule of reuse is the granule of release. "*
- The Common Closure Principle (CCP). *"Classes that change together belong together. "*
- The Common Reuse Principle (CRP). *"Classes that aren 't reused together should not be grouped together. "*

3.5 Validation

We evaluated the theorized design as PoC, and validated by both testing ourselves and getting opinions from experts in the fields. The PoC we have is well tested on a multi-cloud environment, and stress tested. Also the whole idea, design, and the PoC got praise and valuable feedback from a Research Lab and a few industry practitioners

Chapter 4

Results or findings

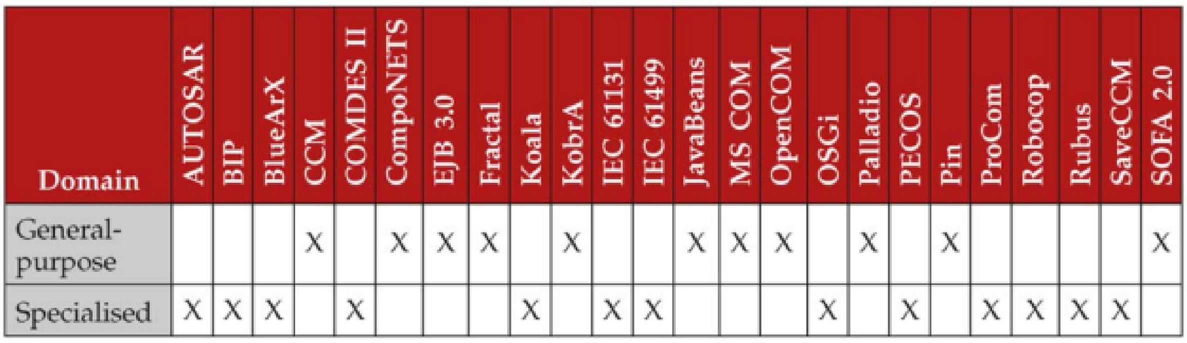
The key findings and innovations we're proposing are:

- A Component Model
- A distributed platform for Components
- Evaluation of the existing Component Models, their pros & cons
- The cost of mainstream FaaS and Microservice approach
- The value of Language virtualization, combined with CBSE
- The end-to-end design of the whole system, findings that helped optimize it, and PoC [41]

The Component Model:

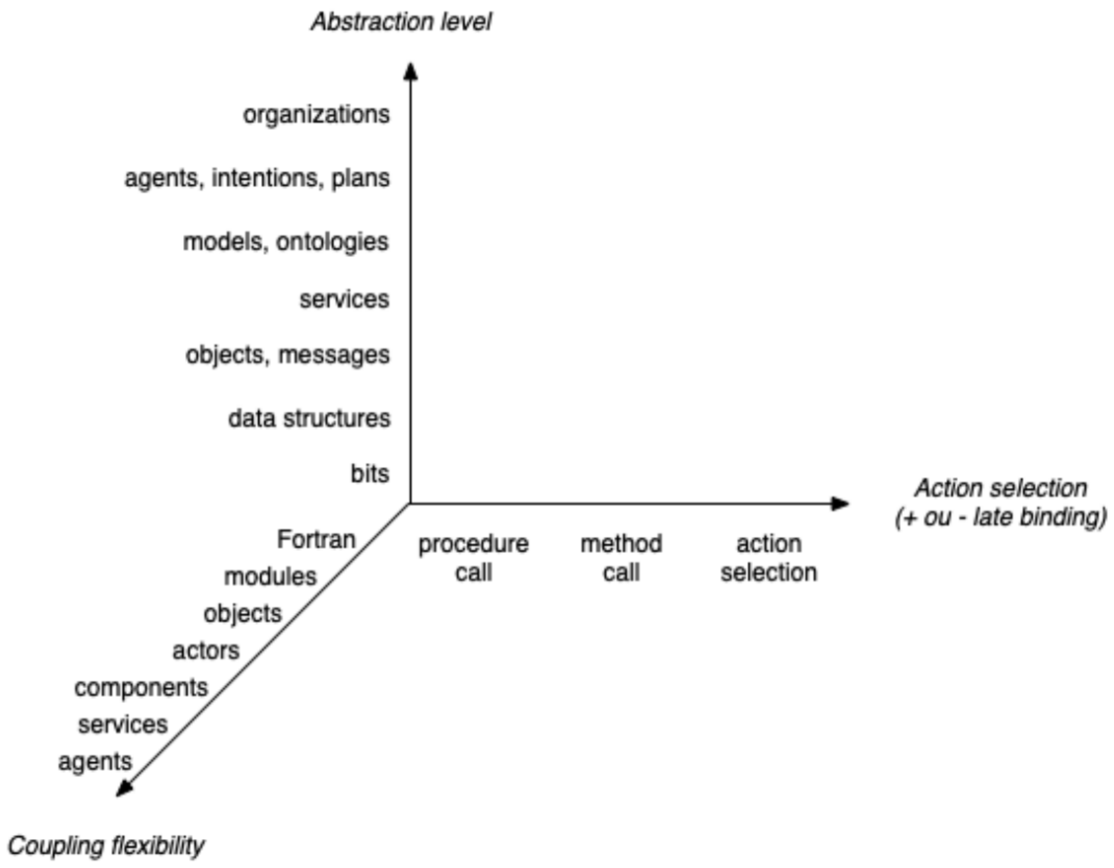
As there's no widely adopted definition for what is "Component", we came up with our own definition, to make the term a bit less ambiguous, and be of practical usage. One of the major criterias an artifact must conform to be called a Component is that it must have a Component Model.

There're a lot of Component Models invented and used in practice till to this day. Some of them are specialized for specific use-case / platform / language ecosystems, and some are generalized.



Domain	AUTOSAR	BIP	BlueArX	CCM	COMDES II	CompoNETS	EJB 3.0	Fractal	Koala	KobrA	IEC 61131	IEC 61499	JavaBeans	MS COM	OpenCOM	OSGi	Palladio	PECOS	Pin	ProCom	Robocop	Rubus	SaveCCM	SOFA 2.0
General-purpose				X		X	X	X		X			X	X	X		X		X					X
Specialised	X	X	X		X				X		X	X				X		X		X	X	X	X	

Component Model is a concern of Programming In-The-Large (PITL), and should abstract away any concerns of Programming In The Small (PITS). That implies, a Component should act like blackbox, and the Component developer may use any paradigms sensible to program the computation logic. A Component should also have some level of autonomy.



Components may change its behavior based on its circumstances. The topology of Component Composition should be flexible, not fixated by the initial wiring. This enables components to be more general for usage as the kernel for more abstract paradigms like agents - or more precisely "System" in General Systems Theory.

We propose a Component Model that has all the desired properties mentioned.

In this model, there's no inherent notion of hierarchy in itself, but gives the programmer the ability to form any arbitrary topology, by reifying two core concepts: Port and Connector. A component only has access to its own ports, and connectors.

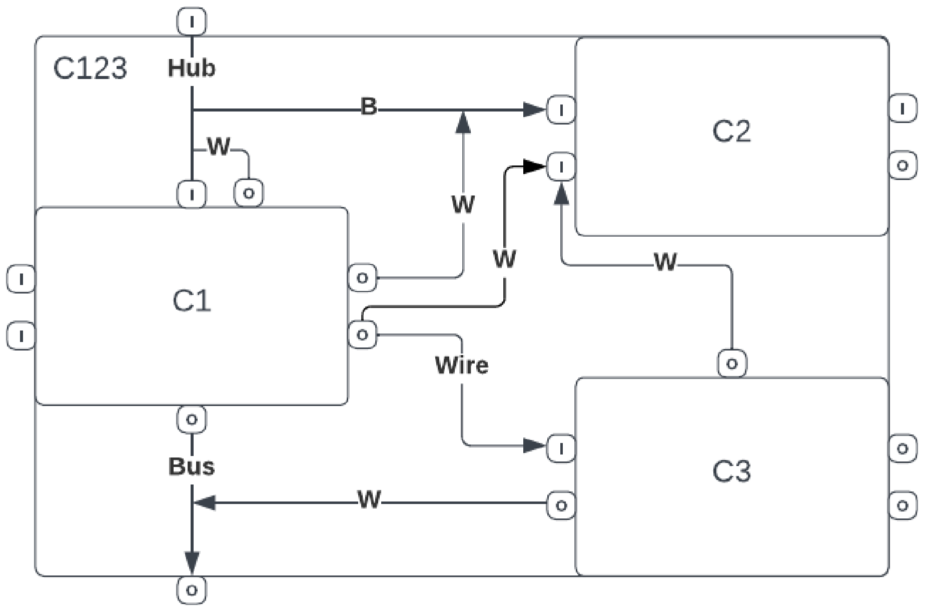
Ports

- Reified form of Component Interfaces.
- Unique and can't be formed arbitrarily without the platform's acknowledgement.
- One way buffer for the events.
- Can be created, or deleted anytime in a Components life-cycle only by itself and for itself

- May be shared with other components via messages

Connectors

- Analogous to wires in PCBs.
- They're the key ingredient for systems dynamics.
- Can be of different types (e.g., wire, bus, hub) with different ordering semantics and delivery guarantee.
- Are part of components internal logic.



In the diagram, even though it seems like C1,C2,C3 Components are inside the C123 Component, it's only the analytical model. C1-3 components are not children of C123, but instead C123 is the composition of the other components. C1-3 may become part of some other composition, and it's the programmers decision whether it should be allowed. This flexibility still does not sacrifice any of the values of Components, and is aligned with the original vision of Alan-Kay.

This opinionated Component Model for our proposed platform (described next) also acts like a wrapper for existing applications written in FaaS, Microservice or Legacy paradigms. Also, the implementation of the Components can be swapped anytime, that is written in a different language, or be updated - without downtime.

The Platform

For Components to truly become of practical use, and become a thing that can be easily adopted - there's a need for a platform, or more like an distributed operating system for the components.

The platform needs to take care of a few things mentioned below:

- Manage the life-cycle of the Components.
- Schedule Components on heterogeneous hardware and environment.
- Resource Provisioning (memory, storage, cpu, gpu, etc.)
- Support applications that can't be embedded in the in-process runtime.

That implies, the platform needs a few components:

- Component Registry
- Distributed scheduler for in-process or exo-process components.
- Executor - the runtime for the components code

The platform needs to have some infrastructure, and compose existing platform level tools. Also, an Internal Developer Platform (IDP) for the developers of the platform itself. These are all relevant for CompaaS, and no off-the-shelf solution exists for our specific architecture needs.

GraalVM

Graal is a key component for the platform we're proposing. It's a Language Virtualization technology, originally built for usage in Oracle database for embedded function execution in any language. By utilizing Partial Evaluation approach and JVM optimizations, graal achieves more than 3x performance gain than the official runtime for a language.

In our case, a significant value proposition GraalVM polyglot API gives us is that we can eliminate inter-component communication latency, along with the overhead of containers for each small component - that results in compactness and efficiency by a great margin.

The native image capability of graal can further reduce the memory usage and reduce warmup time of JVM. With Graal Isolates, we can ensure security on the tenant level, or even on the basis of developers' needs.

Scala/Akka

Akka is an Actor Toolkit for building highly scalable, distributed systems. It has most of the building blocks that we need for the platform to schedule/passivate Components. Also, with Aeron, Akka can achieve great performance for inter-node communication.

Internal Development Platform (IDP) and Infrastructure:

Although the components are managed by the platform (CompaaS), there has to be some infrastructure to scale the platform itself. For that, we evaluated 100+ tools/options from CNCF landscape, and picked the best option that matches our architecture goal.

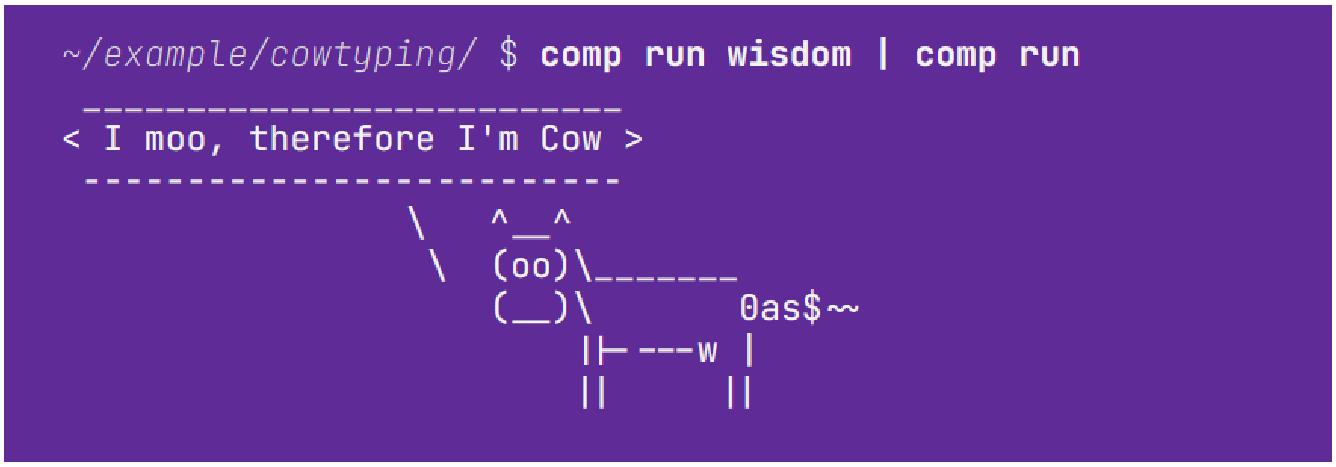
Here's a short list of them, omitting the comparison for brevity:

- **Container Orchestration**: Kubernetes
- **K8s Distro**: K3s (for prod) and Minikube (for dev) or Managed (e.g., AKS, OKE, GKE)
- **Control Plane**: Crossplane
- **CI/CD Orchestration**: Keptn
- **Continuous Delivery**: ArgoCD
- **Continuous Integration**: Github-actions
- **Load Testing**: K6
- **Monitoring**: Grafana
- **Log Collector**: Loki
- **Telemetry**: Prometheus, OpenTelemetry, Jaeger
- **Application Definition & Image Build**: KubeVela (OAM), Tilt, Docker BuildX
- **Database**: ScyllaDB

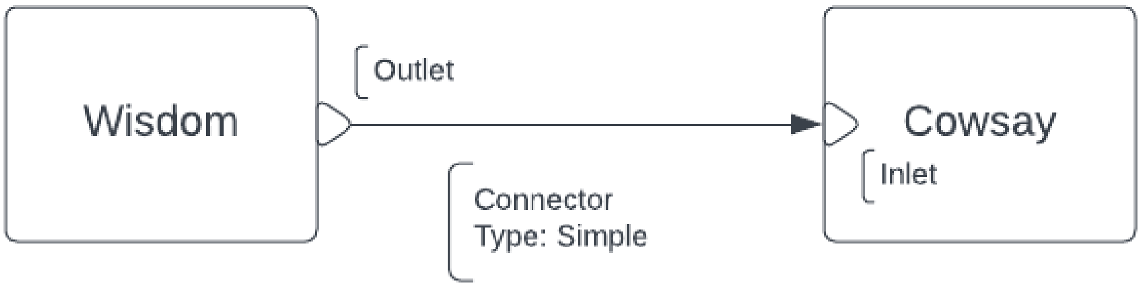
Proof-of-Concept & Findings:

Example - CowTyping

In this simple example, there’s two components called “Wisdom” and “Cowsay”
Wisdom - Emits pseudo-wisdom, and written in Javascript.
Cowsay - Emits ASCII-Art of a cow saying the message it received, comically, and written with Rust, Compiled to WebAssembly.

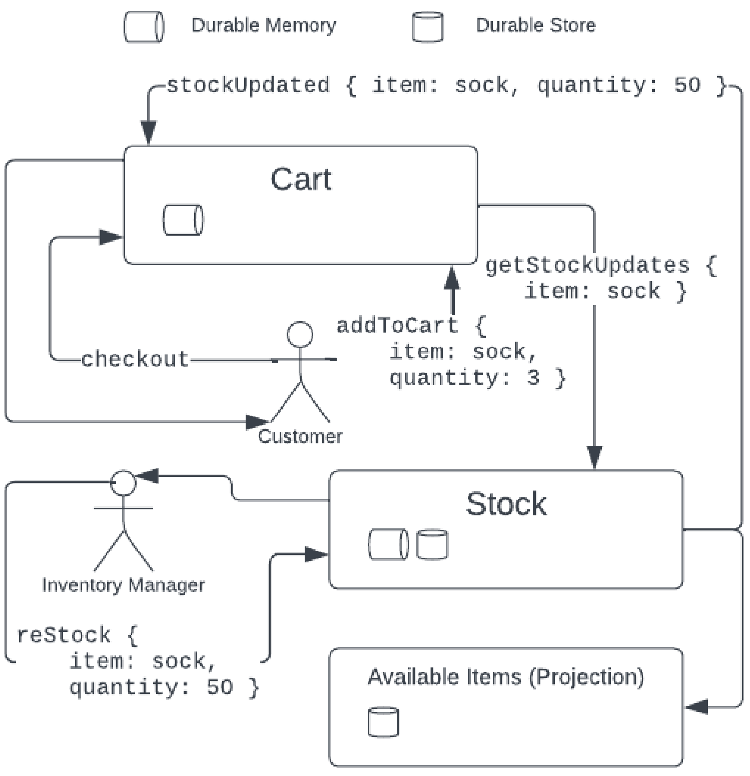


```
~/example/cowtyping/ $ comp run wisdom | comp run
 __________________________
< I moo, therefore I'm Cow >
 --------------------------
                 \   ^__^
                  \  (oo)\_______
                     (__)\           0as$~~
                         ||----w |
                         ||     ||
```



Example - Shopping Cart

Cart - stores item SKU, along with quantity, for individual Customer
Stock - tracks availability of all items
Available Items - receives stock updates, and maintains a list of available products
Durable Memory - directly accessible in-memory storage
Durable Store - for arbitrary size, and semantic data



Durable Memory
Durable Store
stockUpdated { item: sock, quantity: 50 }
Cart
getStockUpdates { item: sock }
addToCart { item: sock, quantity: 3 }
checkout
Customer
Stock
Inventory Manager
reStock { item: sock, quantity: 50 }
Available Items (Projection)

Benchmarks

We've done some necessary measurement of the overhead of Akka, compared to plain execution on java threads. Also, the throughput difference of component implementation in wasm and javascript. We used JMH, with a warmup of 10 iteration, and 5 measurement iteration.

Benchmark	Language	Score	Error
IncBenchmark.invoke	js	106.558	3.846
IncBenchmark.invoke	wasm	194.427	9.341
NthPrimeBenchmark.invoke	js	1968984.972	62740.956
NthPrimeBenchmark.invoke	wasm	751792.852	90302.402
SlugifyBenchmark.invoke	js	13325.471	742.388
SlugifyBenchmark.invoke	wasm	9297.969	3888.442
akka.IncBenchmark.inc	js	179.107	8.218
akka.IncBenchmark.inc	wasm	151.479	11.636
akka.NthPrimeBenchmark.nthPrime	js	498586.980	13408.664
akka.NthPrimeBenchmark.nthPrime	wasm	161354.230	7418.734
akka.SlugifyBenchmark.slugify	js	3357.406	244.308
akka.SlugifyBenchmark.slugify	wasm	508.264	44.810

Here, IncBenchmark, SlugifyBenchmark, and NthPrimeBenchmark are 3 different components (listed in increasing cpu usage requirement)

From the table, we can conclude that Akka adds a little bit of overhead over simple threads, but by better scheduling of components on cores (1 thread per core), it achieves better results overall.
If we use thread-local graal polyglot context, and re-use them for each language, throughput and memory usage improves significantly. To know whether this has any downsides, we reached core developers of graal, and with their feedback, we've also taken necessary measures.

Chapter 5

Discussion

CompaaS is designed from ground-up in a first-principle thinking approach - asking questions in every step.

The component model we're proposing aligns with our goal to make components similar to Universals. By reification of ports and connectors, the model gives a scope to incorporate capability based security and extreme late-bindings. This extreme late-binding blurs away the dichotomy of design time and runtime aspect of software development. It's also necessary for higher kinds of abstractions, like Agents.

In the microservice approach, there's a strong emphasis on not sharing databases with multiple services. But as the services get more granular and increase in number, the cost of isolated databases for each service becomes problematic. Also, the cost of containerizing each service including their runtime and dependency has a massive impact on efficiency. Our model solves both problems by secure isolation of components with very low overhead, and compacted in a single JVM instance on each node.

In contrast to the Function as a Service approach, our components can be stateful, have multiple interfaces, run in the same process and share data to other components in very minimal or no serialization overhead. Stateful functions can have state associated with each function acting like a local variable - and we are inspired by implementations of this approach like in Flink Statefun, and gathered valuable insights for our purpose.

Actor Model is one of the major sources of inspiration for us. The Component Composition scheme is similar to what gul-agha described in his paper as "Actor System". Components and Actors have common properties like, they both have their internal state, spawn new instances, have queues for incoming/outgoing messages, and act as a unit of concurrency. But the semantics differ in a subtle way for the communication aspect. Components can not directly send messages to another component. Direct messaging creates coupling that gets hard to tame.

It should be explicitly mentioned that, CompaaS is not a replacement of FaaS or Microservice, but a platform for them. It’s one of the parts of a larger scope. We had to narrow down the scope of the thesis to the minimum.

Chapter 6

Conclusion

This research is part of a long journey. The outcome of the whole process can be divided into 3 parts: Learnings, Findings, Innovations. And it covers all the criterias to say that we've met our initial goal for the thesis.

Learnings

In short, we learned how to:

- Take an observation (seemingly problematic phenomenon);
- Analyze further the situation with proper context;
- Communicate with experts and validate our subjective opinion;
- Search for solution that might be in theorized form but not applied in practice;
- Find problems in them, and extract the value propositions of the ideas / approaches
- Connect the dots
- Find gap in the graph of knowledge
- Work on to fill that gap
- Validate if the new work is consistent with that graph by doing end-to-end proof of concept
- Finally, verify the approach taken, and subjective conclusion by taking experts opinions

Findings

Things that we found by our own experimentation and analysis:

- What are the limitations of GraalVM, performance characteristics and memory usage for each language (tested wasm, js, python, and verified by core Graal dev team)
- Overhead and Usability of Akka, compared to Go-Lang, Erlang, Flink Statefun - for our use-case
- The necessary configurations to use for graal polyglot api, and isolates
- The memory usage of actors in latest Akka release (verified by core Akka dev team)
- Difference in memory / cpu overhead between our proposed approach and FaaS /

Microservice approach
- Validate the Wormhole approach to use Akka with GraalVM
- How to align Akka Actors more to the Gul-Agha's Actor Semantics and Actor System

Innovations

The innovations that we worked on are combined into one unique solution - for a problem identified and described in this document, and named it "CompaaS". CompaaS - acronym for Component as a Service, is a platform that makes granularization of Services practical, a goal that the application of Microservice approach failing to achieve currently in industry.

In concise terms, CompaaS is a unique combination of an opinionated Component Model and a Runtime. It is end-to-end designed to be linearly scalable, utilizing state of the art technologies, and prior research in CBSE. Our unification of System Theoretic approach to Component Model is a new addition to Component Literature in Computer Science.

Limitations

There're a few limitations, and open questions for further research. A few are stated below:

- GraalVM isn't supported on the browser environment yet. So although the design allows the components to be scheduled on heterogeneous compute nodes (e.g., Edge, Mobile, IoT), the browser environment needs some extra work to be included.
- It's still practically not possible to make primitives like the number "1" as a Component, without any change or exception - but it should be, as the number is a system too. Our design currently treats these primitives as given from the platform.
- JVM is not designed to be used for proper utilization of Actor Model, or Thread-Per-Core Architecture, like in GoLang, Erlang.
- Possibility to incorporate CHERI in CompaaS should be considered with great potential value proposition and subject to evaluation.
- Only the PoC is done, but the end to end materialization of our proposed architecture needs some time, effort and deal with the devil in the detail

Bibliography

[1] Feo, J. T., et al. "The Common Component Architecture." Computing in Science & Engineering 4.2 (2002): 38-47.
[2] Alur, D., Crupi, J., & Malks, D. Core J2EE Patterns: Best Practices and Design Strategies. Prentice Hall, 2001.
[3] Kay, A., & Goldberg, A. (1976). Personal dynamic media. ACM SIGPLAN Notices, 11(8), 74-85.
[4] Kay, A. (1993). The early history of Smalltalk. ACM SIGPLAN Notices, 28(3), 69-95.
[5] Philip Wadler - The essence of functional programming https://dl.acm.org/doi/pdf/10.1145/143165.143169
[6] Zhenjiang Hu, John Hughes, Meng Wang - How functional programming mattered https://academic.oup.com/nsr/article/2/3/349/1427872
[7] J Hughes - Generalising monads to arrows source : https://www.sciencedirect.com/science/article/pii/S0167642399000234
[8] Andrzej Filinski - Representing monads . Source : https://dl.acm.org/doi/abs/10.1145/174675.178047
[9] Programming in the Small https://link.springer.com/article/10.1007/s10278-009-9271-z
[10] Programming-in-the-Large Versus Programming-in-the-Small https://ieeexplore.ieee.org/document/1702345/
[11] C. V. Ramamoorthy; Vijay Garg; Atul Prakash - Programming in the large https://ieeexplore.ieee.org/abstract/document/6312978
[12] Programming-in-the-Large Versus Programming-in-the-Small | https://ieeexplore.ieee.org/document/1702345/
[13] Compilers, Techniques, and Tools for Supporting Programming https://ieeexplore.ieee.org/document/6674271/
[14] Su J., Tian J.B. (2019)Microservice Architecture of Web Application in Cloud Environment. Electronic Technology and Software Engineering, (15):131-132.
[15] Brandon B. (2017)Serverless Computing: Next Generation Cloud Infrastructure. Computer world, 2017-05-15(003).
[16] Zhi Yun. (2018)Read the advantages and disadvantages of serverless architecture, and use cases. http://www.sohu.com/a/234002113_100159565.
[17] N. Marz, J. Warren, Big Data: Principles and best practices of scalable realtime data systems. Manning Publications, 2013.
[18] Fowler, M. (2005). Event Sourcing. Retrieved from https://martinfowler.com/eaaDev/EventSourcing.html
[19] Young, G. (2010). CQRS and Event Sourcing. Retrieved from https://www.codeproject.com/Articles/555855/CQRS-Event-Sourcing

[20] Young, G. (2010). Event Sourcing and CQRS: A Journey. Retrieved from https://cqrs.files.wordpress.com/2010/11/cqrs_documents.pdf
[21] Zdun, P., & Schulte, S. (2015). Event Sourcing: Foundations and Trends in Databases. Foundations and Trends in Databases, 6(1-2), 1-93.
[22] Luckham, D. C. (2002). The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley Professional.
[23] Luckham, D. C. (2008). Event Processing for Business: Organizing the Real-Time Enterprise. John Wiley & Sons.
[24] Etzion, O., & Niblett, P. (2010). Event Processing: Designing IT Systems for Agile Companies. McGraw-Hill Education.
[25] Chen, L., & Hsu, C. (2014). Complex Event Processing: A Survey. ACM Computing Surveys (CSUR), 46(3), 1-42.
[26] Abadi, D., et al. (2005). The Design of the Borealis Stream Processing Engine. In CIDR (pp. 277-289).
[27] V. S. Pai, P. Druschel, and W. Zwaenepoel, "Flash: An efficient and portable web server," in Proceedings of the Annual Conference on USENIX Annual Technical Conference, ser. ATEC '99. Berkeley, CA, USA: USENIX Association, 1999, pp. 15–15. [Online]. Available: http://dl.acm.org/citation.cfm?id=1268708.1268723
[28] M. Welsh, D. Culler, and E. Brewer, "SEDA: An Architecture for Well-conditioned, Scalable Internet Services," in Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles, ser. SOSP '01. New York, NY, USA: ACM, 2001, pp. 230–243. [Online]. Available: http://doi.acm.org/10.1145/502034.502057
[29] http://www.seastar-project.org/
[30] Carl Hewitt, John Woods - Inconsistency Robustness https://philpapers.org/rec/HEWIR
[31] Carl Hewitt- Inconsistency Robustness for Logic Programs 0904.3036v33.pdf (arxiv.org)
[32] David S. Olson,Stefania Fusco - Rules versus Standards: Competing Notions of Inconsistency Robustness in Patent Law
[33] https://www.oreilly.com/library/view/cloud-native-infrastructure/9781491984291/ch01.html
[34] https://www.cherryservers.com/blog/the-complete-overview-of-devops-cloud-native-tools#orchestration--management
[35] https://thenewstack.io/cloud-native/the-cloud-native-landscape-observability-and-analysis/
[36] Chafi, Hassan & DeVito, Zach & Moors, Adriaan & Rompf, Tiark & Sujeeth, Arvind & Hanrahan, Pat & Odersky, Martin & Olukotun, Kunle. (2010). Language Virtualization for Heterogeneous Parallel Computing," ser. Onward. ACM SIGPLAN Notices. 45. 835-847. 10.1145/1932682.1869527.
[37] Mogensen, T.Æ. (1999). Partial Evaluation: Concepts and Applications. In: Hatcliff, J., Mogensen, T.Æ., Thiemann, P. (eds) Partial Evaluation. DIKU 1998. Lecture Notes in Computer Science, vol 1706. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-47018-2_1
[38] https://www.itu.dk/people/sestoft/pebook/jonesgomardsestoft-a4.pdf
[39] https://www.oracle.com/in/java/graalvm/what-is-graalvm/
[40] https://firecracker-microvm.github.io/
[41] https://github.com/audacioustux/compaas
[42] https://alo.dev/

[43] Clemens Szyperski, Dominik Gruntz, Stephan Murer (2002). Component Software: Beyond Object-Oriented Programming. 2nd ed. ACM Press - Pearson Educational, London 2002 ISBN 0-201-74572-0
[44] White Paper: Five Challenges That Hinder Microservices (intel.com) - https://www.intel.com/content/dam/www/central-libraries/us/en/documents/2022-10/five-challenges-to-microservices-white-paper.pdf
[45] ACTORS: A Model of Concurrent Computation in Distributed Systems (mit.edu) - https://dspace.mit.edu/handle/1721.1/6952
[46] Actor Model of Computation: Scalable Robust Information Systems (arxiv.org) - https://arxiv.org/abs/1008.1459

Appendix A

Example Component Code

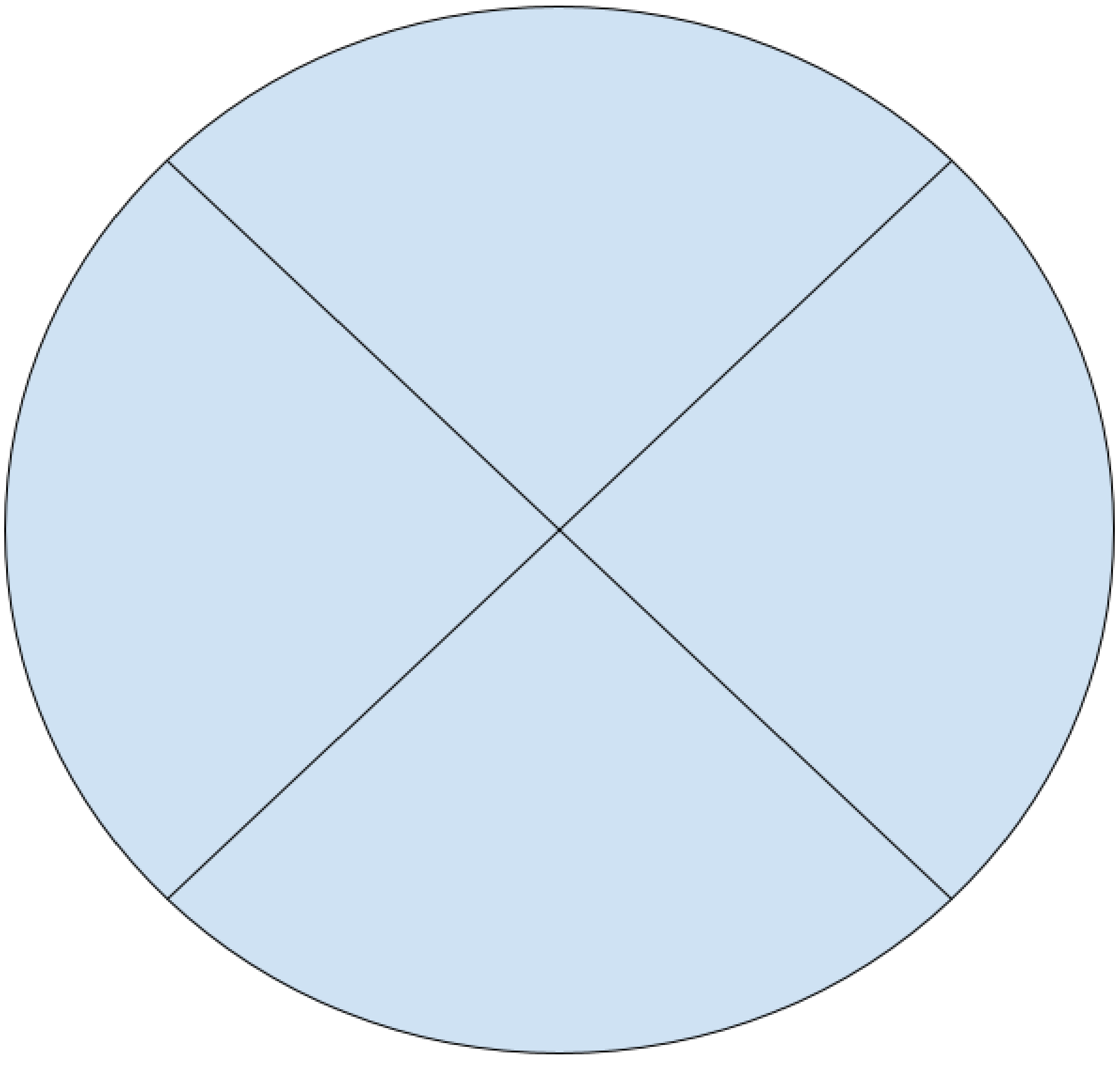
A.1 Increment Counter - JS

```
// emits event on ports: twice - if 2x, thrice - if 3x)

export default ( msg, state, ports ) => {
    // state is managed by the platform transparently
    // no extra sdk is needed
    const { count = 0 } = state
    count++

    return () => {
         // only run when state mutations is persisted
         // state can’t be mutated in this scope
         const { twice, thrice } = ports

         count % 2 || twice.push(count)
         count % 3 || thrice.push(count)
    }
}
```

Appendix B

Design Drafts

B.1 Component Model - Design Phase



Appendix C

Resources, Discussions - Drafts

Component-based Software Engineering

1. Szyperski, C. (2002). Component Software. Addison-Wesley
2. https://docs.spring.io/spring-framework/docs/current/reference/html/core.html
   Spring's Beans component model
3. https://www.omg.org/spec/CCM/4.0/PDF
   CORBA Component Model - a major component model which we used to use, for example to develop Cisco communicator app's backend integrations.
4. ActiveX - Wikipedia - https://en.wikipedia.org/wiki/ActiveX
   Microsoft's component model - one of the major ones in the industry at that time.
5. JavaBeans - Wikipedia - https://en.wikipedia.org/wiki/JavaBeans
   Java's original component model
6. Jakarta Enterprise Beans - Wikipedia - https://en.wikipedia.org/wiki/Jakarta_Enterprise_Beans
   Enterprise Beans, which is still active and now being developed with a fresh thrust by Jakarta EE.
   https://jakarta.ee/specifications/enterprise-beans/4.0/
   We can refer to it as the Enterprise Beans component model.
   https://jakarta.ee/specifications/enterprise-beans/4.0/jakarta-enterprise-beans-spec-core-4.0.html
7. OSGi The Eclipse Foundation - https://www.osgi.org/
   Akka in OSGi • Akka Documentation -
   https://doc.akka.io/docs/akka/current/additional/osgi.html
8. The very first paper proposing "Software Components"-
   https://www.cs.dartmouth.edu/~doug/components.txt
9. WebAssembly/component-model: Repository for design and specification of the Component Model - https://github.com/WebAssembly/component-model
   "A [WebAssembly] component would have a text and binary representation, symmetric to core [wasm] modules. The definition of the text and binary representation would embed core modules' text and binary representations (just like the currently-proposed module-linking binary format sketch, except the outer module is a component, not another core module). Components would describe their interface in terms of interface types used in the types of import and export definitions (just like core modules do today, but replacing core valtype with intertype). Thus, no separate manifest or IDL would be needed at load-/run-time."
10. https://en.wikipedia.org/wiki/Composability

11. https://www.amazon.com/Objects-Components-Frameworks-UML-Catalysis/dp/0201310120/ref=sr_1_1?dchild=1&keywords=objects+components+frameworks+with+uml&qid=1628926469&sr=8-1
    This is the D'Souza book on components (and the Szyperski book), based on which I had presented the seminar on component-based software engineering and how components should be composed

Containers and its limitations

1. https://opencontainers.org/
   This is the industry standard for containers now, not Docker.

   Bloom will be able to run OCI containers via Krustlet or any standard k8s node.
2. https://cri-o.io/
   CRI is the de facto industry standard spec for running containers in k8s. In theory, we can include any CRI runtime to run standard containers in Bloom.
   With CRI runtime, you can run OCI containers. That's the standard.

3. https://kubernetes.io/docs/setup/production-environment/container-runtimes/
   Docker is no longer the preferred runtime for running containers in k8s and it is no longer the preferred way to generate container images for k8s.

Stateful FaaS and its problems

1. https://www.youtube.com/watch?v=G0I8InYbrUk&feature=youtu.be&ab_channel=FlinkForward
2. https://github.com/apache/flink-statefun

Actor Model

1. https://arxiv.org/pdf/1008.1459.pdf
2. Actor Model of Computation for Scalable Robust Information Systems - Archive ouverte HAL - https://hal.archives-ouvertes.fr/hal-01163534v7
3. https://aeplay.org/citybound
   One of the coolest demos for what can be achieved with lightweight actor model programming
4. Hewitt, Meijer and Szyperski: The Actor Model (everything you wanted to know, but were afraid to ask) | Going Deep | Channel 9 - https://channel9.msdn.com/Shows/Going+Deep/Hewitt-Meijer-and-Szyperski-The-Actor-Model-everything-you-wanted-to-know-but-were-afraid-to-ask
5. Hewitt, Meijer and Szyperski: The Actor Model (everything you wanted to know, but were afraid to ask) | Going Deep | Channel 9 - https://channel9.msdn.com/Shows/Going+Deep/Hewitt-Meijer-and-Szyperski-The-Actor-Model-everything-you-wanted-to-know-but-were-afraid-to-ask
6. Gul Agha's ActorFoundry actor lib for Java - https://osl.cs.illinois.edu/software/actor-foundry/tutorial.pdf

Scala

1. https://www.youtube.com/playlist?list=PLajEH1jD0zp-RhavpKZf9pxI5XZL12NLB

His talks were presented in multiple confs, so you can watch those too by searching and mainly the Q&A part of the talks

2. https://www.coursera.org/learn/effective-scala?aid=true
   https://scala-lang.org/blog/2021/05/17/effective-programming-scala-course.html
   Official Scala course updated for Scala 3
   Gradually complete this course
3. https://scalabridge.gitbook.io/curriculum/
4. From First Principles: Why Scala? - https://www.lihaoyi.com/post/FromFirstPrinciplesWhyScala.html #must-read
5. Scala Beyond 3.0 - The Quest for Simplicity - Martin Odersky - YouTube - https://www.youtube.com/watch?v=NXTjnowBx-c
   Note at 43:40, he is talking about a research direction they are taking where they are trying to use the concept of Capabilities to manage “Effects” on “Resources”.

   That’s basically what we already have in STEPS in a more elegant and expressive manner!
6. Scala 3 has landed - Martin Odersky - YouTube - https://www.youtube.com/watch?v=JcLG9Ss9Y-w
   “State Monads would be too awkward and inefficient (in Scala compiler).” - Martin Odersky
7. The Essence of Scala | The Scala Programming Language - https://www.scala-lang.org/blog/2016/02/03/essence-of-scala.html
8. Overview | Scala 3 Language Reference | Scala Documentation - https://docs.scala-lang.org/scala3/reference/overview.html
9. Learn Scala with Online Courses | The Scala Programming Language - https://www.scala-lang.org/blog/2022/01/19/learn-scala-with-online-courses.html
10. Sustainable Scala | The Scala Programming Language - https://www.scala-lang.org/blog/2021/12/14/sustainable-scala.html
11. https://www.youtube.com/watch?v=0yRAtLL18cY&ab_channel=EntropyOfSoftware

Akka

1. https://www.youtube.com/playlist?list=PLajEH1jD0zp8MuGOVLWOCgijaZIXJoGha
2. https://www.youtube.com/playlist?list=PLajEH1jD0zp9_wxtnbp2-VQH6EnLYk1aU
3. https://www.youtube.com/playlist?list=PLajEH1jD0zp9vvWMTE6OGRpxPR9rqlfq-
4. https://blog-en.richardimaoka.net/dispatcher-behavior
   Watch this animation at 0.25x to get an overview of how exactly a message is dispatched and an actor gets scheduled.
5. https://doc.akka.io/docs/akka/current/index.html
6. Modularity, Composition and Hierarchy • Akka Documentation - https://doc.akka.io/docs/akka/current/stream/stream-composition.html
7. https://doc.akka.io/docs/akka/current/typed/interaction-patterns.html

GraalVM

1. Youtube Playlist :
   https://www.youtube.com/playlist?list=PLajEH1jD0zp8kyJ_OqCuN_w_HZnVTrRWK
   After watching the Graal playlist I had shared earlier, please start reading the Graal official content and the blog - you will learn lots of exciting capabilities of Graal:
   https://www.graalvm.org/
   https://medium.com/graalvm

2. https://www.graalvm.org/reference-manual/embed-languages/
3. http://teavm.org/
4. https://www.graalvm.org/reference-manual/polyglot-programming
5. https://www.graalvm.org/reference-manual/embed-languages/#build-native-images-from-polyglot-applications
   The GraalVM Polyglot API allows code caching across multiple contexts. Code caching allows compiled code to be reused and allows sources to be parsed only once. Code caching can often reduce memory consumption and warm-up time of the application.
   This is how the interpreted code is cached as compiled code after first run

Papers read:

1. Agha, G.: Actors: a Model of Concurrent Computation in Distributed Systems. Series in Artificial Intelligence. MIT Press (1986)
2. Allen, R., Garlan, D.: Formal connectors. Research Report CMU-CS-94-115, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA (1994)
3. Beugnard, A., J´ez´equel, J.M., Plouzeau, N., Watkins, D.: Making components contract aware. IEEE Computer 32(7), 38–45 (1999)
4. Bigus, J., Schlosnagle, D., Pilgrim, J., Mills, W., Diao, Y.: Able: A toolkit for building multiagent autonomic systems. IBM Systems Journal 41(3), 350–371 (2002)
5. Boissier, O.: (editor) Composants et syst`emes multi-agents. L'Objet 12(4) (2006)
6. Bordini, R., Dastani, M., Dix, J., Seghrouchni, A.E.F., Gomez-Sanz, J., Leite, J., O'Hare, G., Pokahr, A., Ricci, A.: A survey of programming languages and platforms for multi-agent systems. Informatica 30(1), 33–44 (2006)
7. Briot, J.P.: Mod´elisation et classification de langages de programmation concurrente `a objets : l'exp´erience Actalk. In: Actes du Colloque Langages et Mod`eles `a Objets (LMO 94), pp. 153–165. INRIA/IMAG/PRC-IA, Grenoble, France (1994)
8. Briot, J.P.: An experiment in classification and specialization of synchronization schemes. In: K. Futatsugi, S. Matsuoka (eds.) Object Technologies for Advanced Software (ISOTAS 96), no. 1049 in LNCS, pp. 227–249. Springer, Kanazawa, Japan (1996)
9. Briot, J.P.: Composants et agents : ´evolution de la programmation et analyse comparative. Technique et Science Informatiques (TSI) 33(1-2), 85–115 (2014)
10. Briot, J.P., Cointe, P.: A uniform model for object-oriented languages using the class abstraction. In: J. McDermott (ed.) 10th International Joint Conference on Artificial Intelligence (IJCAI'87), vol. 1, pp. 40–43. Morgan-Kaufmann, Milano, Italy (1987)
11. Briot, J.P., Guerraoui, R., L¨ohr, K.P.: Concurrency and distribution in object-oriented programming. Computing Surveys 30(3), 291–329 (1998)
12. Briot, J.P., Meurisse, T., Peschanski, F.: Architectural design of component-based agents: A behavior-based approach. In: R.H. Bordini, M. Dastani, J. Dix, A.E.F. Seghrouchni (eds.) Programming Multi-Agent Systems - ProMAS 2006, no. 4411 in LNCS, pp. 73–92. Springer (2007)
13. Bruneton, E., Coupaye, T., Leclerc, M., Qu´ema, V., Stefani, J.B.: An open component model and its support in Java. In: 7th International Symposium on Component-Based Software Engineering, no. 3054 in LNCS, pp. 7–22. Springer (2004)
14. Caromel, D.: Toward a method of object-oriented concurrent programming. Communications of the ACM (CACM) 36(9), 90–102 (1993)

15. Castagna, G.: Covariance and contravariance: Conflict without a cause. ACM Transactions on Programming Languages and Systems (TOPLAS) 17, 431–447 (1995)
16. Chauvet, J.M.: Services Web avec SOAP, WSDL, UDDI, et XML. Eyrolles (2002)
17. Chen, G., Kotz, D.: A survey of context-aware mobile computing research. Technical Report TR2000-381, Department of Computer Science, Dartmouth College, Hanover, NH, USA (2000)
18. Cointe, P.: Metaclasses are first class: The ObjVlisp model. In: Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications (OOPSLA '87), pp. 156–162. ACM (1987)
19. Cointe, P. (ed.): Meta-Level Architectures and Reflection – Second International Conference, Reflection'99 Saint-Malo, France, July 19-21, 1999 Proceedings. No. 1616 in LNCS. Springer (1999)
20. Coutaz, J., Crowley, J.L., Dobson, S., Garlan, D.: Context is key. Communications of the ACM 48(3), 49–53 (2005)
21. Crnkovi´c, I., Sentilles, S., Vulgarakis, A., Chaudron, M.R.V.: A classification framework for software component models. IEEE Transactions on Software Engineering 37(5), 593–615 (2011)
22. Cutsem, T.V., Dedecker, J., Mostinckx, S., Gonzalez, E., D'Hondt, T., Meuter, W.D.: Ambient references: Addressing objects in mobile networks. In: OOPSLA '06: Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications, pp. 986–997. ACM (2006)
23. Drogoul, A., Collinot, A.: Applying an agent-oriented methodology to the design of artificial organisations: a case study in robotic soccer. Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS) 1(1), 113–129 (1998)
24. Dubus, J., Merle, P.: Vers l'auto-adaptabilit´e des architectures logicielles dans les environnements ouverts distribu´es. In: M. Oussalah, F. Oquendo (eds.) 1`ere Conf´erence francophone sur les Architectures Logicielles (CAL 2006). Herm`es/Lavoisier, Nantes, France (2006)
25. Eugster, P., Felber, P., Guerraoui, R., Kermarrec, A.M.: The many faces of publish/subscribe. ACM Computing Surveys 35(2), 114–131 (2003)
26. Ferber, J., Gutknecht, O.: A meta-model for the analysis and design of organizations in multi-agent systems. In: 3rd International Conference on Multi-Agent Systems (ICMAS 98), pp. 128–135. IEEE, Paris, France (1998)
27. Finin, T., Labrou, Y., Mayfield, J.: KQML as an agent communication language. In: J. Bradshaw (ed.) Software Agents, pp. 291–316. MIT-Press (1997)
28. FIPA: Agent Communication Language Specifications (accessed: 13/08/2021). http://www.fipa.org/repository/aclspecs.html
29. FIPA: Foundation for Intelligent Physical Agents (accessed: 13/08/2021). http://www.fipa.org/
30. Gasser, L., Briot, J.P.: Object-based concurrent programming and distributed artificial intelligence. In: N.M. Avouris, L. Gasser (eds.) Distributed Artificial Intelligence: Theory and Praxis, pp. 81–107. Kluwer (1992)
31. Gasser, L., Briot, J.P.: Agents and concurrent objects. IEEE Concurrency 6(4), 74–81 (1998). Interview of Les Gasser by Jean-Pierre Briot
32. Gelernter, D., Carrierro, D.: Coordination languages and their significance. Communications of the ACM 35(2) (1992)
33. Georgeff, M., Pell, B., Pollack, M., Tambe, M., Wooldridge, M.: The belief-desireintention model of agency. In: 5th International Workshop on Intelligent Agents V:

Agent Theories, Architectures, and Languages (ATAL'98), no. 1555 in LNCS, pp. 1–10.
Springer (1999)
34. Ghezzi, C., Picco, G.: An outlook on software engineering for modern distributed
systems. In: Monterey Workshop on Radical Approaches to Software Engineering.
Venezia, Italy (2002)
35. Hewitt, C.: Viewing control structures as patterns of passing messages. Artificial
Intelligence 8(3), 323–364 (1977)
36. H¨ubner, J.F., Sichman, J.S., Boissier, O.: Developing organised multiagent systems
using the MOISE+ model: programming issues at the system and agent levels. International Journal of
Agent-Oriented Software Engineering (IJAOSE) 1(3–4), 370–395
(2007)
37. RoboCup Federation Inc: RoboCup (accessed: 13/08/2021).
https://www.robocup.org/
38. Janson, P.A.: Dynamic linking and environment initialization in a multi-domain process. ACM
SIGOPS Operating Systems Review 9(5), 43–50 (1975)
39. Kafura, D., Briot, J.P.: Introduction to actors and agents. IEEE Concurrency 6(2),
24–29 (1998)
40. Kiczales, G., des Rivieres, J., Bobrow, D.G.: The Art of the Metaobject Protocol. MIT
Press (1991)
41. Lau, K.K., Wang, Z.: Software component models. IEEE Transactions on Software
Engineering 33(10), 709–724 (2007)
42. Meyer, B.: Object-Oriented Software Construction. Prentice Hall (1988)
43. Meyer, B.: Applying design by contract. IEEE Computer 25(10), 40–51 (1992)
44. M¨uller, J.P., Pischel, M.: The agent architecture InteRRaP: Concept and application.
Technical Report RR-93-26, DFKI, Saarbr¨ucken, Germany (1993)
45. OASIS: Open composite services architecture (CSA). Tech. rep., OASIS (Organization for the
Advancement of Structured Information Standards), http://www.oasisopencsa.org (accessed:
13/08/2021)
46. OASIS: Web services business process execution language (BPEL). Tech. rep.,
OASIS (Organization for the Advancement of Structured Information Standards),
http://bpel.xml.org (accessed: 13/08/2021)
47. Odell, J.: Objects and agents compared. Journal of Object Technology (JOT) 1(1)
(2002)
48. Odersky, M., Spoon, L., Venners, B.: Programming in Scala. Artima (2010)
49. OMG: Common object request broker architecture (CORBA). Tech. rep., Object
Management Group (OMG), http://www.omg.org/corba/ (accessed: 13/08/2021)
50. OMG: Corba component model (CCM). Tech. rep., Object Management Group
(OMG), http://www.omg.org/technology/documents/formal/components.htm (accessed: 13/08/2021)
51. OMG: Model driven architecture (MDA). Tech. rep., Object Management Group
(OMG), http://www.omg.org/mda/ (accessed: 13/08/2021)
52. Omicini, A., Ricci, A., Viroli, M.: Artifacts in the A&A meta-model for multi-agent
systems. Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS) 17(3),
432–456 (2008)
53. Papazoglou, M.: Web Services & SOA, Principles and Technology, Second Edition.
Pearson (2012)
54. Payne, T.: Web services from an agent perspective. IEEE Intelligent Systems 23(2),
12–14 (2008)
55. Perrot, J.F.: Objets, classes et h´eritage : D´efinitions. In: R. Ducournau, J. Euzenat,

G. Masini, A. Napoli (eds.) Langages et mod`eles `a objets – Etat des recherches et ´
perspectives, Collection Didactique, pp. 3–31. INRIA (1998)
56. Shaw, M., Garlan, D.: Software Architectures – Perspective on an Emerging Discipline.
Prentice Hall (1996)
57. Shoham, Y.: Agent oriented programming. Artificial Intelligence 60(1), 51–92 (1993)
58. Silva, V., Choren, R., Lucena, C.: Using UML 2.0 activity diagram to model agent
plans and actions. In: International Conference on Autonomous Agents and MultiAgent Systems
(AAMAS 2005). Utrecht, The Netherlands (2005)
59. Smith, R.: The contract net protocol: High-level communication and control in a distributed problem
solver. IEEE Transactions on Computers 29(12), 1104–1113 (1980)
60. van Splunter, S., Wijngaards, N., Brazier, F., Richards, D.: Automated componentbased
configuration: Promises and fallacies. In: AISB 2004 Convention 4th Symposium
on Adaptive Agents and Multi-Agent Systems (AAMAS-4), pp. 130–145. Leeds, UK
(2004)
61. Sun: Javabeans specification. Tech. rep., Sun Microsystems Inc.,
http://java.sun.com/products/javabeans/ (2006)
62. Szyperski, C., Gruntz, D., Murer, S.: Component software: beyond object-oriented
programming. Pearson Education (2002)
63. W3C: World Wide Web Consortium (accessed: 13/08/2021). https://www.w3.org/
64. Weyns, D., Parunak, H.V.D., Michel, F., Holvoet, T., Ferber, J.: Environments for
multiagent systems – state-of-the-art and research challenges. In: D. Weyns, H.V.D.
Parunak, F. Michel (eds.) Environments for Multi-Agent Systems – First International
Workshop, E4MAS 2004, New York, NY, July 19, 2004, Revised Selected Papers, no.
3374 in LNAI, pp. 1–47. Springer (2005)
65. Yonezawa, A., Briot, J.P., Shibayama, E.: Object-oriented concurrent programming in
ABCL/1. Sigplan Notices 21(11), 258–268 (1986). Special Issue. Proceedings of the
Conference on Object-Oriented Programming Systems, Languages and Applications
(OOPSLA 86), Portland OR, USA
66. Ziemke, T., Balkenius, C., Hallam, J. (eds.): From Animals to Animats 12 – 12th
International Conference on Simulation of Adaptive Behavior, SAB 2012, Odense,
Denmark, August 2012, Proceedings. No. 7426 in LNAI. Springer (2012)

----

[1] B. Meyer, “What to Compose,” Software Development Magazine, Beyond Objects column,
http://www.ddj.com/architect/ 184414588, Mar. 2000.
[2] C. Szyperski, Component Software: Beyond Object-Oriented Program- ming. Addison-Wesley
Professional, Dec. 1997.
[3] G.T. Heineman and W.T. Councill, Component-Based Software Engineering: Putting the Pieces
Together. Addison-Wesley Longman Publishing Co., 2001.
[4] N. Medvidovic, E.M. Dashofy, and R.N. Taylor, “Moving Architectural Description from under the
Technology Lamppost,” Information and Software Technology, vol. 49, no. 1, pp. 12-31, 2007.
[5] N. Medvidovic and R.N. Taylor, “A Classification and Compar- ison Framework for Software
Architecture Description Lan- guages,” IEEE Trans. Software Eng., vol. 26, no. 1, pp. 70-93, Jan.
2000.
[6] M.R.V. Chaudron and I. Crnkovic, Software Engineering: Principles and Practice, third ed., ch. 18.
Wiley, 2008.

[7] I. Crnkovic, M.R.V. Chaudron, and S. Larsson, “Component- Based Development Process and Component Lifecycle,” J. Computing and Information Technology, vol. 13, no. 4, pp. 321- 327, Nov. 2005.
[8] I. Crnkovic and M. Larsson, Building Reliable Component-Based Software Systems. Artech House, Inc., 2002.
[9] A. Beugnard, J.-M. Je´ ze´ quel, N. Plouzeau, and D. Watkins, “Making Components Contract Aware,” Computer, vol. 32, no. 7, pp. 38-45, 1999.
[10] I. Crnkovic, M. Larsson, and O. Preiss, “Concerning Predictability in Dependable Component-Based Systems: Classification of Quality Attributes,” Architecting Dependable Systems III, pp. 257- 278, 2005.
[11] G. Kotonya, I. Sommerville, and S. Hall, “Towards a Classification Model for Component-Based Software Engineering Research,” Proc. 29th EUROMICRO Conf., pp. 43-52, 2003.
[12] K.-K. Lau and Z. Wang, “Software Component Models,” IEEE Trans. Software Eng., vol. 33, no. 10, pp. 709-724, Oct. 2007.
[13] M. Akerholm, J. Carlson, J. Fredriksson, H. Hansson, J. Ha˚kans- son, A. Mo¨ller, P. Pettersson, and M. Tivoli, “The SAVE Approach to Component-Based Development of Vehicular Systems,” J. Systems and Software, vol. 80, no. 5, pp. 655-667, May 2007.
[14] S. Sentilles, A. Vulgarakis, T. Bures, J. Carlson, and I. Crnkovic, “A Component Model for Control-Intensive Distributed Embedded Systems,” Proc. 11th Int’l Symp. Component Based Software Eng., pp. 310-317, Oct. 2008.
[15] H. Maaskant, A Robust Component Model for Consumer Electronic Products, vol. 3, pp. 167-192. Springer, 2005.
[16] J. Cheesman and J. Daniels, UML Components: A Simple Process for Specifying Component-Based Software. Addison-Wesley Longman Publishing Co., Inc., 2000.
[17] R. van Ommering, F. van der Linden, J. Kramer, and J. Magee, “The Koala Component Model for Consumer Electronics Soft- ware,” Computer, vol. 33, no. 3, pp. 78-85, 2000.
[18] S. Becker, H. Koziolek, and R. Reussner, “Model-Based Perfor- mance Prediction with the Palladio Component Model,” Proc. Sixth Int’l Workshop Software and Performance, pp. 54-65, 2007.
[19] C. Atkinson, J. Bayer, C. Bunse, E. Kamsties, O. Laitenberger, R. Laqua, D. Muthig, B. Paech, J. Wu¨st, and J. Zettel, Component-Based Product Line Engineering with UML. Addison-Wesley Longman Publishing Co., Inc., 2002.
[20] S. Hissam, J. Ivers, D. Plakosh, and K.C. Wallnau, “Pin Component Technology (V1.0) and Its C Interface,” Technical Note: CMU/SEI- 2005-TN-001. www.sei.cmu.edu/pub/documents/05.reports/pdf/05tn001.pdf, Apr. 2005.

[21] E.E. Group, “JSR 220: Enterprise JavaBeansTM,Version 3.0 EJB Core Contracts and Requirements Version 3.0, Final Release,” May 2006.
[22] “OMG CORBA Component Model v4.0,” http://www.omg.org/ spec/CCM/4.0/, 2011.
[23] D. Box, Essential COM. Object Technology Series. Addison-Wesley, 1997.
[24] Oxford Advanced Learners Dictionary, http://www.oxfordadvanced learnersdictionary.com/, 2011.
[25] Sun Microsystems, “Javabeans Specification,” java.sun.com/ javase/technologies/desktop/javabeans/docs/spec.html, 1997.
[26] OSGi Alliance, “OSGi Service Platform Core Specification, V4.1,” 2007.
[27] T.O.M. Group, “UML Superstructure Specification v2.1,” http:// www.omg.org/spec/UML/2.1.2/, Apr. 2009.
[28] IEC, “Application and Implementation of IEC 61131-3,” IEC, 1995. [29] K. Hanninen, J. Maki-Turja, M. Nolin, M. Lindberg, J. Lundback, and K. Lundback, “The Rubus Component Model for Resource Constrained Real-Time Systems,” Proc. Int’l Symp. Industrial

Embedded Systems, pp. 177-183, June 2008.
[30] E. Bruneton, T. Coupaye, and J. Stefani, “The Fractal Component
Model Specification,” The ObjectWeb Consortium, technical report,
http://fractal.objectweb.org/specification/index. html, Feb. 2004.
[31] A. Basu, M. Bozga, and J. Sifakis, “Modeling Heterogeneous Real- Time Components in Bip,” Proc. Fourth IEEE Int’l Conf. Software Eng. and Formal Methods, pp. 3-12, 2006.
[32] W. Emmerich, M. Aoyama, and J. Sventek, “The Impact of Research on the Development of Middleware Technology,” ACM Trans. Software Eng. and Methodology, vol. 17, no. 4, pp. 1-48, 2008.
[33] M. Shaw, “Truth vs Knowledge: The Difference between What a Component Does and What We Know It Does,” Proc. Int’l Workshop on Software Specification and Design, pp. 181-185, 1996.
[34] “PECT Homepage,” www.sei.cmu.edu/pacc/pect_init.html, ac- cessed July 2008.
[35] S. Sentilles, P. Stepan, J. Carlson, and I. Crnkovic, “Integration of Extra-Functional Properties in Component Models,” Proc. 12th Int’l Symp. Component Based Software Eng., June 2009.
[36] S.J. Mellor and M. Balcer, Executable UML: A Foundation for Model- Driven Architectures. Addison-Wesley Longman Publishing Co., Inc., 2002.
[37] H. Breivold and M. Larsson, “Component-Based and Service- Oriented Software Engineering: Key Concepts and Principles,” Proc. EUROMICRO Conf. Software Eng. and Advanced Applications, pp. 13-20, Aug. 2007.
[38] AUTOSAR Development Partnership, “AUTOSAR—Technical Overview V2.0.1,” www.autosar.org, June 2006.
[39] J.E. Kim, O. Rogalla, S. Kramer, and A. Haman, “Extracting, Specifying and Predicting Software System Properties in Compo- nent Based Real-Time Embedded Software Development,” Proc. 31st Int’l Conf. Software Eng., 2009.
[40] X. Ke, K. Sierszecki, and C. Angelov, “COMDES-II: A Component- Based Framework for Generative Development of Distributed Real-Time Control Systems,” Proc. 13th IEEE Int’l Conf. Embedded and Real-Time Computing Systems and Applications, pp. 199-208, 2007.
[41] R. Bastide and E. Barboni, “Component-Based Behavioural Modelling with High-Level Petri Nets,” Proc. Third Workshop Modelling of Objects, Components and Agents, pp. 37-46, Oct. 2004.
[42] IEC, “IEC 61499 Function Blocks for Embedded and Distributed Control Systems Design,” IEC, 2005.
[43] M. Clarke, G. Blair, G. Coulson, and N. Parlavantzas, “An Efficient Component Model for the Construction of Adaptive Middle- ware,” Proc. IFIP/ACM Int’l Conf. Distributed Systems Platforms, 2001.
[44] M. Winter, C. Zeidler, and C. Stich, “The PECOS Software Process,” Proc. Workshop Components-Based Software Development Processes, 2002.
[45] J. Muskens, M.R.V. Chaudron, and J.J. Lukkien, “A Component Framework for Consumer Electronics Middleware,” Component- Based Software Development for Embedded Systems, pp. 164-184, Springer, 2005.
[46] T. Bures, P. Hnetynka, and F. Pla ́ sil, “SOFA 2.0: Balancing Advanced Features in a Hierarchical Component Model,” Proc. Int’l Conf. Software Eng. Research, Management and Applications, pp. 40-48, 2006.

[47] J. Feljan, L. Lednicki, J. Maras, A. Petricic, and I. Crnkovic, “Classification and Survey of Component Models,” Technical Report MRTC report ISSN 1404-3041 ISRN MDH-MRTC-242/ 2009-1-SE, http://www.mrtc.mdh.se/index.php?choice= publications&id=2099, Dec. 2009.
[48] S. Yacoub, H. Ammar, and A. Mili, “A Model for Classifying Component Interfaces,” Proc. Second Int’l Workshop on Component- Based Software Eng. in Conjunction with the 21st Int’l Conf. Software Eng., pp. 17-18, 1999.

[49] S. Yacoub, H. Ammar, and A. Mili, “Characterizing a Software Component,” Proc. Second Workshop on Component-Based Software Eng. in Conjunction with Int’l Conf. Software Eng., 1999.
[50] K.J. Fellner and K. Turowski, “Classification Framework for Business Components,” Proc. 33rd Hawaii Int’l Conf. System Sciences, vol. 8, p. 8047, 2000.
[51] M. Bozga, S. Graf, I. Ober, I. Ober, and J. Sifakis, “The IF Toolset,” Formal Methods for the Design of Real-Time Systems, Int’l School on Formal Methods for the Design of Computer, Comm., and Software Systems, SFM-RT 2004, pp. 237-267, 2004.
[52] G. Go ̈ssler, “Prometheus—A Compositional Modeling Tool for Real-Time Systems.” Proc. Workshop Real-Time System Tools, 2001.
[53] C. Seceleanu, A. Vulgarakis, and P. Pettersson, “REMES: A
Resource Model for Embedded Systems,” Proc. 14th IEEE Int’l Conf. Eng. Complex Computer Systems, June 2009.

------

[1] F. Bachmann, L. Bass, C. Buhman, S. Comella-Dorda, F. Long, J. Robert, R. Seacord, and K. Wallnau, “Volume II: Technical Concepts of Component-Based Software Engineering,” second ed., Technical Report CMU/SEI-2000-TR-008, Software Eng. Inst., Carnegie Mellon Univ., 2000.
[2] Component-Based Software Engineering: Putting the Pieces Together, G. Heineman and W. Councill, eds. Addison-Wesley, 2001.
[3] C. Szyperski, D. Gruntz, and S. Murer, Component Software: Beyond
Object-Oriented Programming, second ed. Addison-Wesley, 2002.
[4] K.-K. Lau, “Software Component Models,” Proc. 28th Int’l Conf.
Software Eng. (ICSE ’06), pp. 1081-1082, 2006.
[5] K.-K. Lau and Z. Wang, A Survey of Software Component Models, second ed., School of Computer Science, Univ. of Manchester, http://www.cs.man.ac.uk/cspreprints/PrePrints/cspp38.pdf, May 2006.
[6] M. Shaw and D. Garlan, Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall, 1996.
[7] L. Bass, P. Clements, and R. Kazman, Software Architecture in Practice, second ed.
Addison-Wesley, 2003.
[8] L. DeMichiel, L. Yalc ̧inalp, and S. Krishnan, Enterprise JavaBeans Specification Version 2.0, 2001.
[9] R. Monson-Haefel, Enterprise JavaBeans, fourth ed. O’Reilly & Assoc., 2004.
[10] P. Clements, “A Survey of Architecture Description Languages,” Proc. Eighth Int’l Workshop Software Specification and Design (IWSSD ’96), pp. 16-25, 1996.
[11] N. Medvidovic and R.N. Taylor, “A Classification and Compar- ison Framework for Software Architecture Description Lan- guages,” IEEE Trans. Software Eng., vol. 26, no. 1, pp. 70-93, Jan. 2000.
[12] K.-K. Lau and Z. Wang, “A Taxonomy of Software Component Models,” Proc. 31st Euromicro Conf. Software Eng. and Advanced Applications (SEAA ’05), pp. 88-95, 2005.
[13] M. Broy, A. Deimel, J. Henn, K. Koskimies, F. Plasil, G. Pomberger, W. Pree, M. Stal, and C. Szyperski, “What Char- acterizes a Software Component?” Software—Concepts and Tools, vol. 19, no. 1, pp. 49-56, 1998.
[14] B. Meyer, “The Grand Challenge of Trusted Components,” Proc. 25th Int’l Conf. Software Eng. (ICSE ’03), pp. 660-667, 2003.
[15] D. Box, Essential COM. Addison-Wesley, 1998.
[16] CORBA Component Model, V3.0, OMG, http://www.omg.org/

technology/documents/formal/components.htm, 2002.
[17] JavaBeans Specification. Sun Microsystems, http://java.sun.com/
products/javabeans/docs/spec.html, 1997.
[18] A. Wigley, M. Sutton, R. MacLeod, R. Burbidge, and S. Wheel-
wright, Microsoft .NET Compact Framework (Core Reference). Micro-
soft Press, Jan. 2003.
[19] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, Web Services:
Concepts, Architectures and Applications. Springer-Verlag, 2004. [20] R. van Ommering, F. van der Linden, J. Kramer, and J. Magee, "The Koala Component Model for Consumer Electronics Soft-
ware," Computer, vol. 33, no. 3, pp. 78-85, Mar. 2000.
[21] R. van Ommering, "The Koala Component Model," Building Reliable Component-Based Software Systems, I. Crnkovic and
M. Larsson, eds., pp. 223-236, Artech House, 2002.
[22] C. Atkinson, J. Bayer, C. Bunse, E. Kamsties, O. Laitenberger, R. Laqua, D. Muthig, B. Paech, J. Wu¨st, and J. Zettel, Component-Based
Product Line Engineering with UML. Addison-Wesley, 2001.
[23] F. Pla´sil, D. Balek, and R. Janecek, "SOFA/DCUP: Architecture for Component Trading and Dynamic Updating," Proc. Fourth Int'l Conf. Configurable Distributed Systems (ICCDS '98), pp. 43-52, 1998. [24] F. Pla´sil, M. Besta, and S. Visnovsky, "Bounding Component Behavior via Protocols," Proc. Technology of Object-Oriented
Languages and Systems (TOOLS 31), pp. 387-398, 1999.
[25] UML 2.0 Superstructure Specification, OMG, http://www.omg.org/
cgi-bin/doc?ptc/2003-08-02, 2007.
[26] J. Cheesman and J. Daniels, UML Components: A Simple Process for
Specifying Component-Based Software. Addison-Wesley, 2000.
[27] T. Genssler, A. Christoph, B. Schulz, M. Winter, C. Stich, C. Zeidler, P. Mu¨ller, A. Stelter, O. Nierstrasz, S. Ducasse, G. Are´valo, R. Wuyts, P. Liang, B. Scho¨nhage, and R. van den Born, PECOS in a Nutshell, http://www.pecos-project.org/, Sept. 2002. [28] O. Nierstrasz, G. Are´valo, S. Ducasse, R. Wuyts, A. Black, P. Mu¨ller, C. Zeidler, T. Genssler, and R. van den Born, "A Component Model for Field Devices," Proc. First Int'l IFIP/ACM
Working Conf. Component Deployment (CD '02), pp. 200-209, 2002. [29] "The Fractal Project Web Page," http://fractal.objectweb.org/,
2007.
[30] E. Bruneton, T. Coupaye, and J. Stefani, "Recursive and Dynamic
Software Composition with Sharing," Proc. Seventh Int'l Workshop
Component-Oriented Programming (WCOP '02), 2002.
[31] E. Bruneton, T. Coupaye, and J. Stefani, "The Fractal Component Model," ObjectWeb Consortium, Technical Report Specification
V2, 2003.
[32] E. Bruneton, T. Coupaye, and M. Leclercq, "An Open Component
Model and Its Support in Java," Proc. Seventh Int'l Symp. Component-Based Software Eng. (CBSE '04), pp. 7-22, 2004.
[33] JavaBeans Architecture: BDK Download. Sun Microsystems, http://
java.sun.com/products/javabeans/software/bdk_download. html, 2003.
[34] Java 2 Platform, Enterprise Edition Sun Microsystems, http:// java.sun.com/j2ee/, 2007.
[35] Common Object Request Broker Architecture: Core Specification, Version 3.0.3,
http://www.omg.org/technology/documents/ corba_spec_catalog.htm, Mar. 2004.

[36] D. Garlan, R. Monroe, and D. Wile, “Acme: Architectural Description of Component-Based Systems,” Foundations of Compo- nent-Based Systems, G. Leavens and M. Sitaraman, eds., pp. 47-68, Cambridge Univ. Press, 2000.
[37] C. Peltz, “Web Services Orchestration and Choreography,” Computer, vol. 36, no. 10, pp. 46-52, Oct. 2003.
[38] M. Lumpe, F. Achermann, and O. Nierstrasz, “A Formal Language for Composition,” Foundations of Component Based Systems, G. Leavens and M. Sitaraman, eds., pp. 69-90, Cambridge Univ. Press, 2000.
[39] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weeragwarana, Business Process Execution Language for Web Services (BPEL4WS) Version 1.1. IBM, http://www-106.ibm.com/ developerworks/library/ws-bpel/, 2004.
[40] R. Allen and D. Garlan, “A Formal Basis for Architectural Connection,” ACM Trans. Software Eng. and Methodology, vol. 6, no. 3, pp. 213-249, 1997.
[41] I. Crnkovic, H. Schmidt, J. Stafford, and K. Wallnau Proc. Sixth Workshop Component-Based Software Eng.: Automated Reason- ing and Prediction (CBSE ’04). ACM SIGSOFT Software Eng. Notes, vol. 29, no. 3, pp. 1-7, May 2004.
[42] B. Christiansson, L. Jakobsson, and I. Crnkovic, “CBD Process,” Building Reliable Component-Based Software Systems, I. Crnkovic and M. Larsson, eds., pp. 89-113, Artech House, 2002.
[43] B. Warboys, B. Snowdon, R. Greenwood, W. Seet, I. Robertson, R. Morrison, D. Balasubramaniam, G. Kirby, and K. Mickan, “An Active Architecture Approach to COTS Integration,” IEEE Soft- ware, special issue on incorporating COTS into the development process, vol. 22, no. 4, pp. 20-27, July/Aug. 2005.
[44] The Bean Builder, Sun Microsystems, https://bean-builder.dev. java.net/, 2007.
[45] AcmeStudio 2.1 User Manual, Carnegie Mellon Univ., http://www-2.cs.cmu.edu/~acme/Manual/AcmeStudio-2.1.htm, 1998.
[46] J. Aldrich, C. Chambers, and D. Notkin, “ArchJava: Connecting Software Architecture to Implementation,” Proc. 24th Int’l Conf. Software Eng. (ICSE ’02), pp. 187-197, 2002.
[47] J. Aldrich, C. Chambers, and D. Notkin, “Architectural Reasoning in ArchJava,” Proc. 16th European Conf. Object-Oriented Program- ming (ECOOP ’02), pp. 334-367, 2002.
[48] J. Aldrich, D. Garlan, B. Schmerl, and T. Tseng, “Modeling and Implementing Software Architecture with Acme and ArchJava,” Proc. Companion 19th Ann. ACM SIGPLAN Conf. Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA ’04), pp. 156-157, 2004.
[49] L. DeMichiel and M. Keith, Enterprise JavaBeans, Version 3.0. Sun Microsystems, 2006.
[50] R. Monson-Haefel, Enterprise JavaBeans 3.0, fifth ed. O’Reilly & Associates, 2006.
[51] “Eclipse Web Page,” http://www.eclipse.org/, 2007.
[52] Y. Choi, O. Kwon, and G. Shin, “An Approach to Composition of EJB Components Using C2 Style,” Proc. 28th Euromicro Conf.
(Euromicro ’02), pp. 40-46, 2002.
[53] P. Clements and L. Northrop, Software Product Lines: Practices and
Patterns. Addison Wesley, 2001.
[54] P. Hnetynka and F. Pla´sil, “Dynamic Reconfiguration and Access
to Services in Hierarchical Component Models,” Proc. Ninth Int’l Symp. Component-Based Software Eng. (CBSE ’06), I. Gorton et al., eds., pp. 352-359, 2006.

[55] K.-K. Lau, P. Velasco Elizondo, and Z. Wang, “Exogenous Connectors for Software Components,” Proc. Eighth Int’l Symp. Component-Based Software Eng. (CBSE ’05), I. Gorton et al., eds., pp. 90-106, 2005.
[56] K.-K. Lau, M. Ornaghi, and Z. Wang, “A Software Component Model and its Preliminary Formalisation,” Proc. Fourth Int’l Symp. Formal Methods for Components and Objects (FMCO ’06), F. de Boer et al., eds., pp. 1-21, 2006.
[57] A. Major, COM IDL and Interface Design. John Wiley & Sons, Feb. 1999.
[58] M. Barnett and W. Schulte, “Runtime Verification of .Net Contracts,” Systems and Software, vol. 65, no. 2003, pp. 199-208, 2003.
[59] BEA Systems et al., “CORBA Components,” Object Management Group, Technical Report orbos/99-02-05, 1999.
[60] R. Natan, CORBA: A Guide to Common Object Request Broker Architecture. McGraw-Hill, 1995.
[61] OpenCCM User’s Guide, ObjectWeb—Open Source Middleware, http://openccm.objectweb.org/doc/0.8.1/user_guide.html, 2007. [62] E. Newcomer, Understanding Web Services: XML, WSDL, SOAP, and
UDDI. Addison-Wesley, 2002.
[63] Ariba, Microsoft, and IBM, Web Services Description Language
(WSDL) Version 1.1., http://www.w3.org/TR/2001/NOTE-wsdl- 20010315, 2001.

“The best way to predict the future is to invent it.”

- Alan Kay (1971 at PARC)