CIDE: An Integrated Development Environment for Microservices

Desheng Liu[#], Hong Zhu[*], Chengzhi Xu[+], Ian Bayley[*], David Lightfoot[*], Mark Green[*] and Peter Marshall[*]

[(#)] Science & Technology on Complex Electronic System Simulation Laboratory, Beijing, China. *Email*: liudsnudt@126.com
[(*)] Dept. of Comp. & Comm. Tech., Oxford Brookes Univ., Oxford, UK. *Email*: hzhu@brookes.ac.uk
[(+)] School of Computer Science, Hubei University of Technology, Wuhan, China. *Email*: xcz911@gmail.com

Abstract – Microservices is a flexible architectural style that has many advantages over the alternative monolithic style. These include better performance and scalability. It is particularly suitable, and widely adopted, for cloud-based applications, because in this architecture a software system consisting of a large suite of services of fine granularity, each running its own process and communicating with the others. However, programming such systems is more complex. In this paper we report on CIDE, an integrated software development environment that helps with this. CIDE supports programming in a novel agent-oriented language called CAOPLE and tests their execution in a cluster environment. We present the architecture of CIDE, discuss its design based on the principles of the DevOps software development methodology, and describe facilities that support continuous testing and seamless integration, two other advantages of Microservices.

***Keywords* – Integrated Software Development Environment (IDE); Microservices; Service-Oriented Architectures; Agent-Oriented Programming; Programming Languages and Tools.**

I. INTRODUCTION

Microservices (MS) is a software architecture style in which a large complex software application is decomposed into many services each of small (hence micro) size [1, 2]. These services can be independently deployed to a cluster of computers and duplicated to achieve load balance and system-performance optimization. Each runs in its own process and interacts with other services through lightweight communication mechanisms. The alternative is the so-called monolithic style, in which number of services is small. MS brings many benefits for engineering service-oriented systems. These include support for team development, continuous testing and seamless integration. It is widely considered to be an effective architecture for cloud computing and service-oriented applications [3].

However, developing a system that consists of a large number of micro-scale services running on a farm of servers imposes grave challenges to software engineering [4, 5]. These include:

- *Program complexity*, due to the thousands of services at micro scale running asynchronously in a distributed computer network. Programs that are difficult to understand are also hard to write and hard to modify.
- *Performance criticality*, especially because MS is often adopted in order to improve performance, and this requires the system to be elastic according to the workload. MS enables services to be duplicated and distributed to a cluster of servers but the choice of where to duplicate and distribute to can have a great impact on performance. We must therefore be able to test systems with different duplication and distribution patterns.
- *Evolution continuity*, which is often required by the application domain for which MS has been chosen. MS enables flexible modification of the component microservices and seamless integration of new functionality into an existing system. However, ensuring this advantage is a difficult task, because the development and testing must be done in a cluster environment.

Our solution to the complexity challenge is a novel programming language called CAOPLE, introduced in our previous work [6] where we demonstrated that it can be used to program service-oriented systems as part of MS. COAPLE is based on the caste-centric conceptual model of multi-agent systems [7]. It has been implemented by compiling high-level source code into object code for a lightweight language-specific virtual machine called CAVM-2, which is a complete redesign of an initial prototype called CAVM. In this paper, we present an integrated software development environment called CIDE (CAOPLE Integrated Development Environment) that supports programming in CAOPLE. Most importantly, it supports the precise control over deployment and testing that is necessary to overcome the performance criticality and evolution continuity challenges mentioned above.

II. OVERVIEW OF CAOPLE LANGUAGE

A. Key Features of CAOPLE Language

CAOPLE stands for Caste-centric Agent-Oriented Programming Language and Environment [7]. Here, agents are service providers that are analogous to real-world counterparts such as estate agents that buy and sell properties, and travel agents that buy and sell air tickets. It is worth noting that agents can themselves be service requesters.

In the literature on service-oriented architectures, including that on MS, the word "service" can mean either the functionality provided by the computer system [8] or the computational entity that provides that functionality. Here, we use the word "service" with only the first meaning, reserving the word "agent" for the second. In our conceptual

978-1-5090-2628-9/16 $31.00 © 2016 IEEE
DOI 10.1109/SCC.2016.112

model, an agent is an autonomous entity running in its own process. Agents encapsulate data, operations and behaviour rules. They execute in parallel and cooperate with each other through a set of well-defined asynchronous communication channels. This corresponds exactly with the second meaning of services.

CAOPLE also introduces the classifier of agents as a language facility, which is called a *caste.* Therefore, agents are runtime instances of castes just like objects are runtime instances of classes. Conversely, a caste is a template for agents just like a class is a template for objects. The word "microservice" can either mean one of several identical copies of a service, each being an agent in our conceptual model, or it can mean a template from which instances can be generated and deployed to different servers, i.e. a caste in our conceptual model. So the concepts of caste and agent that we have just introduced correspond precisely to these two different meanings.

An important difference between caste and class is that an agent can join a caste and quit from a caste dynamically, even suspending and resuming its caste membership at runtime. Moreover, an agent can be a member of multiple castes and a caste can extend multiple castes. The caste facility provides a powerful language facility that supports programming microservices.

CAOPLE is agent-oriented because agents are the basic building blocks of programs. It is in fact purely agent-oriented because the agent is the only building block we have. It is also caste-centric because programs are constructed from castes and every agent must be created as an instance of a caste. Readers are referred to [6] for more details of the language.

B. Implementation of COAPLE on CAVM Virtual Machine

COAPLE is implemented as the compilation of source code into object code of the CAVM-2 virtual machine, which is designed for languages like CAOPLE running in a distributed computing environment. CAVM-2 provides a runtime environment in the same way that Java Virtual Machine (JVM) supports the execution of Java program.

However, CAVM-2 is significantly different from the JVM in both architecture and functionality. CAVM-2 consists of two key elements:

- *Communication Engine* (CE) provides a lightweight communication mechanism for network transparent communications between agents.
- *Local execution engine* (LEE) executes the instructions of the program code and fulfils the computation logic.

These elements can be distributed on each hardware node of a cluster in any combination: CE only, LEE only or both CE and LEE. This is illustrated in Figure 1. Agents can be executed on any LEE and communicate with other agents through the CE without knowing their physical location on the network, so CAOPLE code can be network-transparent. More details of the virtual machine and its implementation can be found in [6].

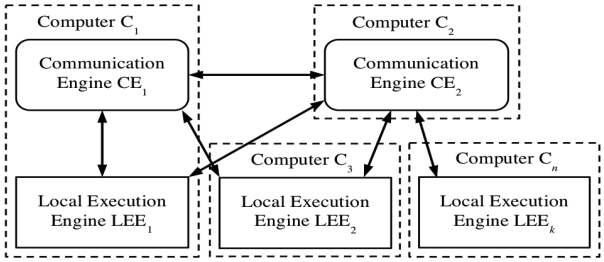

Figure 1 Overall Structure of CAVM Virtual Machine

To achieve optimal performance for the cluster, it is important to allocate carefully the CEs and LEEs to the computer nodes, to deploy and distribute the castes to the CEs and to allocate and distribute agents to various LEEs in the network. As the workload profile of an application changes dynamically, these CE/LEE allocations and caste/agent distributions must be adapted flexibly. The CAVM-2 virtual machine provides a flexible mechanism for such configuration of the cluster and deployment and distribution of agents; see [6] for more details.

However, the challenge that remains is how to turn such a mechanism into a facility that enables the developers to test and debug such a system in various platform configurations and code distributions.

III. The CIDE Development Environment

A. Overall Architecture and Design Principle

CIDE is designed based on two principles. The first is that it should support the caste-centric agent-oriented software development methodology [7]. This methodology deals with the complexity of dynamic multi-agent systems through modelling and formal specification at a high level of abstraction, together with automated tools transforming models and formal specifications into program code.

Therefore, as shown in Figure 2, we integrated into CIDE a tool for constructing graphic models of systems in a modelling language called CAMLE. We also integrated

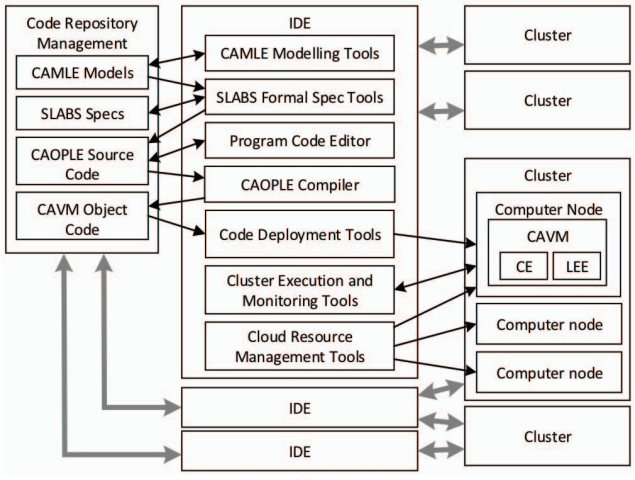

Figure 2 Architecture of CIDE

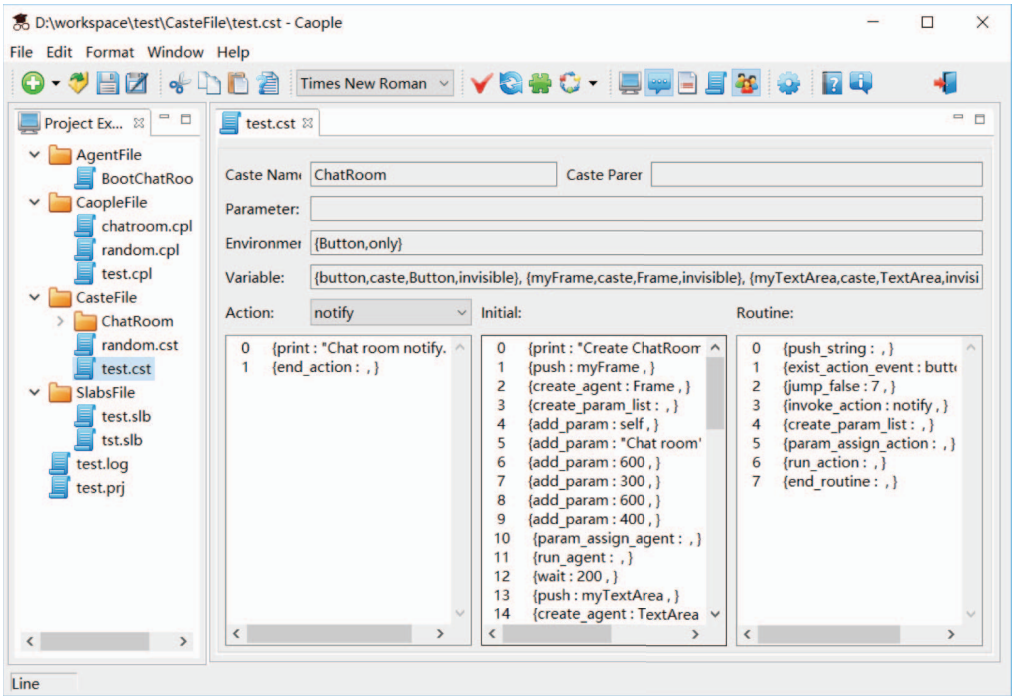

Figure 3 CIDE's Graphic User Interface

tools for checking model consistency and completeness, for transforming CAMLE models into formal specifications in SLABS [9, 10, 11], for editing formal specification of multi-agent systems in SLABS [12], and for automatically transforming formal specifications in SLABS into CAOPLE source code skeletons. Such tools help to reduce the complexity of programming by modelling system structure with graphic models. They also help with the testing, validation and verification of the program against formal specifications.

Figure 3 shows the overall GUI of CIDE, in which various types of software artefacts, including bytecode in the case shown, can be viewed and edited in multiple tabs, and then processed to generate new artefacts with various types of tools.

The second principle on which CIDE is designed is to support evolutionary and agile development, including continuous testing and integration in particular. It followed the so-called "*shift to the left*" principle of DevOps philosophy. In DevOps, engineering activities associated with testing, deployment and operation are shifted towards earlier stages of development (e.g. coding) in order to enable incremental development and seamless integration. Following this principle, we designed and implemented a set of tools and facilities that help to narrow the gap between development and operation of service-oriented systems.

As shown in Figure 2, CIDE also extends the traditional software development environment with three new types of facilities: (a) the management and configuration of cloud resources, (b) the deployment and distribution of object code to a cluster of computers, (c) parallel and distributed execution and monitoring of MS on a cluster.

These facilities provide key support for MS. There are some tools already available on the market that provide similar supports, often as a part of the container technology. However, as far as we know, all the existing tools are operational tools, not IDE plug-ins. Turning existing tools into IDE plug-ins is difficult, if not impossible, because these tools allow only one instance to run at a time to manage a cluster, whereas an IDE is normally used by many developers simultaneously. It is also difficult to design and implement a tool that enables multiple developers to manage and monitor multiple clusters as a part of the development environment. From a developer's point of view, such a cluster could be either a testing facility or a real operation facility. Whichever it is, the tool should treat them in the same way so that test executions can be realistic. The difficulty of managing and configuring the cluster is even greater if the environment includes a complicated software stack. Luckily, CAOPLE programs can be run on a much simpler platform of CAVM-2. CIDE overcomes these difficulties by taking advantage of the existing mechanisms provided by the CAVM-2 virtual machine. The following describes these facilities of CIDE.

B. Management and Configuration of Cloud Resources

The prototype tool for cloud resource management and configuration in CIDE has two main functions. First of all, it can deploy the CAVM-2 virtual machine to any of the nodes in a cluster. It scans the IP addresses in a user-specified range to detect active computer nodes and then it obtains their host names. CAVM-2 can then be deployed on a computer node when authorized remotely through file transmission and remote execution of programs. Secondly, the tool enables the developer to turn on and off the CEs and LEEs on each node of a cluster of computers. In this way, a cluster can be configured by setting which nodes run CEs and are communication-only, which run LEEs, and are computation-only, and which nodes run both, as identifying these can reduce the cross-node communications. Figure 4 shows the graphical user interface for the cloud management tool.

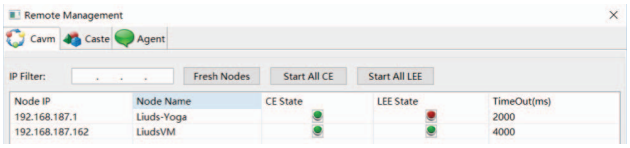

Figure 4 The Cloud Management and Configuration Facility

C. Deployment of Castes to CEs

If a caste is deployed to a CE then that CE manages all instances of the caste (i.e. its agents). In addition, all messages sent to and generated by these agents are through the CE. The object code of the caste is also stored on the computer node and when an agent is instantiated on an LEE, the object code is downloaded to the computer node where the LEE is located, if the code does not already exist on the node.

The deployment tool allows the user to see a list of the castes on a CE, and to manually load the object code of a new caste to the CE. Figure 5 shows its user interface.

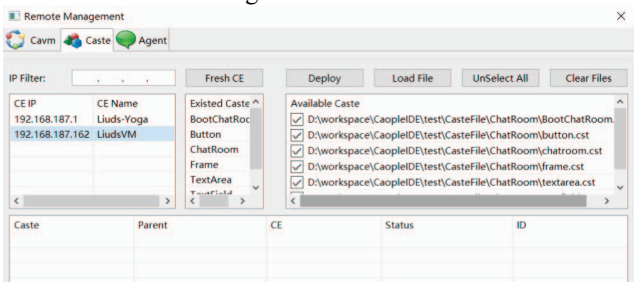

Figure 5. The Caste Deployment Facility

D. Agent Management

The agent management facility allows the user to create and launch agents on an LEE running in the network and to monitor the execution states of agents on a specific LEE. As Figure 6 shows, only a few button clicks are needed to deploy an agent and launch it on any LEE. Its ID and execution state are also displayed.

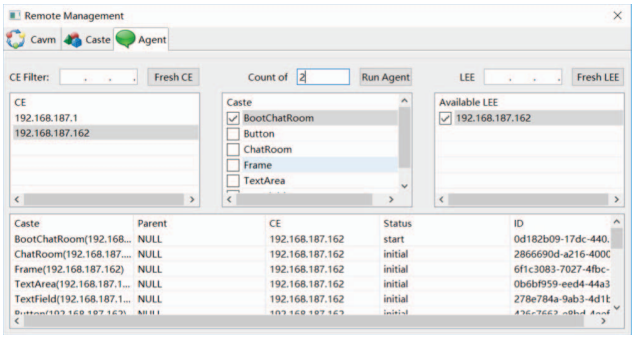

Figure 6. The Agent Management Facility

IV. Conclusion

In this paper, we proposed a new type of integrated software development environment that supports the development of service-oriented applications in MS and we reported the CIDE system that we have been developing for the CAOPLE programming language.

A. Related Work

As cloud computing and big data have become the main trend of recent years, the IT industry urgently needs a new approach to meet the scalability challenges of so-called Big SaaS [13]. Many companies and organizations have adopted MS. These include Amazon, eBay, and Netflix. However, problems remain in how to develop applications in MS and how to improve the efficiency of running a large number of microservices in a cluster of servers. In [6], we identified three main challenges to the development and operation of service-oriented applications in the MS architecture. These are (a) how to program a large set of fine-grained services running in parallel, (b) the need for a lightweight communication facility for the large number of MS to collaborate with each other, and (c) the need for a flexible code deployment mechanism and facility that enable services to be deployed to a cluster easily.

In the past two years, much work has been reported to address challenges (b) and (c) above. Almost all of them adopt the so-called container technology, which is a kind of lightweight virtualization technique. It allows programs to share memory, processors and the file system, but provides separation to the customers [14]. Thousands of containers can be deployed on one host easily and they can restart quickly too. Processes can be deployed to containers flexibly at a cost much less than that of starting a new virtual machine.

Many kinds of containers have emerged. Docker is an open-source project that automates the deployment of applications inside containers by providing an additional layer of abstraction and automation of operating-system-level virtualization on Linux [15]. Many companies have adopted Docker. Amazon has published Container Service (Amazon ECS), which is a highly scalable and fast container management service that makes it easy to run, stop and manage Docker containers on a cluster of Amazon EC2 instances [16]. Google has published an open-source platform, Kubernetes, for automating deployment, scaling and operations of applications on Linux containers across clusters of hosts [17]. Oracle provides Solaris Zones as Container for Oracle Solaris 11 OS [18]. Microsoft has also started to work on the container technology.

However, while container technologies improve the infrastructure on which microservices execute, the programs running in the containers are still programmed in traditional programming languages. Challenge (a) remains an unsolved problem. In fact, the complexity of the programming task for developing MS increases due to the introduction of additional elements of containers and the need to consider runtime efficiency and scalability in a cluster environment.

Our approach proposed in [6] differs from that of container technologies. It advocates a new programming paradigm based on the caste-centric agent-oriented

conceptual model of service-oriented applications. A new programming language called CAOPLE was proposed and implemented. This paper further develops this approach by introducing CIDE, first of a new generation of integrated development environment. It has two distinctive features.

Firstly, it is based on the principles of DevOps software development philosophy. CIDE provides facilities and tools for the developers to manage computational resources in a cluster, to deploy program components to the nodes on a cluster and to create and execute instances of MS on the node. These are the functionalities and facilities that are traditionally provided by system management and monitoring tools and software deployment tools. By integrating them into the IDE, we enable the developers to execute and test their code in a real cluster environment, thereby narrowing the gap between development, deployment, operation and maintenance.

Secondly, CIDE is based on the agent-oriented software development methodology CAMLE for conceptual design of service-oriented systems. In particular, it integrates CAMLE with the agent-oriented modelling language and tools. The graphical models of agent-oriented systems can be automatically checked for their consistency, completeness, and automatically transformed into formal specifications written in SLABS. A formal specification of a service-oriented system written in SLABS can be automatically transformed by CIDE into a CAOPLE code skeleton. Therefore, CIDE not only provides a coding environment, but also supports the complete lifecycle of specification, design, coding, testing and run-time performance tuning of service-oriented applications in a consistent conceptual model.

B. Further Work

CIDE has been implemented in Java and tested on a mini-cluster. More advanced features that provide stronger support to system monitoring, testing and debugging are being developed. In particular, we are revising and adapting a testing automation framework [19] developed for testing multi-agent systems based on SLABS formal specification. Another subject worth further research is the facility for debugging microservices in a distributed parallel computing environment. We are also conducting case studies with the system in the development of programs in CAOPLE with a more powerful cloud infrastructure.

ACKNOWLEDGEMENT

The work reported in this paper is partially supported by EU FP7 project MONICA on Mobile Cloud Computing (Grant No. PIRSES-GA-2011-295222), Oxford Brookes University's Central Research Fund (CRF Phase-2 2015), National Natural Science Foundation of China (Grant No. 61170025), and Natural Science Foundation of Hubei Province, China (Grant No. 2013CFB021).

REFERENCES

[1] Lewis, J., and Fowler, M., "Microservices", URL: http: //martinfowler.com/articles/microservices.html#footnote-monolith, 25 Mar. 2014. (*Last access on 2 Nov. 2015*)

[2] Richardson C., "Introduction to Microservices*"*. URL: https: //www.nginx.com/blog/introduction-to-microservices/, May 19, 2015. (Last access on 2 Nov. 2015)

[3] High Scalability, "The Great Microservices Vs Monolithic Apps Twitter Melee", URL: http://highscalability.com/ blog/2014/7/28/the-great-microservices-vs-monolithic-apps - twitter-melee.html, Jul., 2014. (Last access on 2 Nov. 2015)

[4] NewMan, S., *Building Microservices: Designing Fine-Grained Systems*, O'Reilly, Feb., 2015.

[5] Krause, L., *Microservices: Patterns and Applications*, Amazon.co.uk, Marston Gate, April, 2015.

[6] Xu, C., Zhu, H., Bayley, I., Lightfoot, D., Green, M., and Marshall, P., "CAOPLE: A Programming Language for Microservices SaaS", in *Proc. of SOSE 2016*, pp42-52.

[7] Zhu, H., "Towards An Agent-Oriented Paradigm of Information Systems". In *Handbook of Research on Nature Inspired Computing for Economy and Management*, Jean-Philippe Rennard (Ed), Chapter XLIV, pp679–691, 2006.

[8] Singh, P. M., and Huhns, N. M., *Service-Oriented Computing: Semantics, Processes, Agents*, Wiley, 2005.

[9] Zhu, H. and Shan, L., "Caste-Centric Modelling of Multi-Agent Systems: The CAMLE Modelling Language and Automated Tools", in *Model-Driven Software Development*, Research and Practice in Software Engineering, Vol. II, Beydeda, S. and Gruhn, V. (eds), Springer, 2005, pp57-89.

[10] Shan, L., Du, C., and Zhu, H., "Modeling and Simulating Adaptive Multi-Agent Systems with CAMLE", In *Proc. of COMPSAC 2015*, 1-5 July, 2015, Vol. 2, pp147-152.

[11] Zhu, H. and Shan, L., "Agent-Oriented Modelling and Specification of Web Services", *International Journal of Simulation and Process Modelling*, Vol. 3, No.1&2, pp26 – 44, August, 2007.

[12] Zhu, H., "SLABS: A Formal Specification Language for Agent-Based Systems", *International Journal of Software Engineering and Knowledge Engineering*, Vol. 11. No. 5, pp529-558, Nov., 2001.

[13] Zhu, H., Bayley, I., Younas, M., Lightfoot, D., Yousef, B., Liu, D., "Big SaaS: The Next Step Beyond Big Data", in *Proc. of CLOUD 2015*, pp1131-1140.

[14] Pahl, C., "Containerization and the PaaS Cloud", *IEEE Cloud Computing*, Vol. 2, No. 3, pp24-31, May-Jun. 2015.

[15] Merkel, D., "Docker: Lightweight Linux Containers for Consistent Development and Deployment". *Linux Journal*, Vol. 2014, No. 239, p2, 2014.

[16] Amazon Products & Services, "Amazon EC2 Container Service*"*, URL: https://aws.amazon.com/ecs/. (Last access on 2 Nov. 2015)

[17] Brewer, E. A., "Kubernetes And The Path To Cloud Native", In *Proc. of SoCC'15*, 2015, pp167-167.

[18] Van Surksum, K., "Release: Oracle Solaris 11". URL: http://virtualization.info/en/news/2011/11/release-oracle-solaris-11.html. (Last access on 2 Nov. 2015)

[19] Wang, S., and Zhu, H., "CATest: A Test Automation Framework for Multi-Agent Systems", in *Proc. of IEEE COMPSAC 2012,* July 2012, pp148-157.