A Lightweight Microservice-oriented Platform for Development of Intelligent Agent-based Enterprise Applications

Aluizio Haendchen Filho[1], Rafael Castaneda Ribeiro[3], Hércules Antônio do Prado[2], Edilson Ferneda[2] and Jeferson Miguel Thalheimer[1]
[1]*Laboratory of Technological Innovation in Education (LITE), University of Vale do Itajaí (UNIVALI), Itajaí, Brazil*
[2]*Catholic University of Brasilia (UCB) QS 07, Lote 01, Taguatinga, Brasília, DF, Brazil*
[3]*Federal Center of Technological Education Celso Suckow da Fonseca Rio de Janeiro, RJ, Brazil*

Keywords: Multiagent Systems, Microservice-oriented Development, Service-oriented Development.

Abstract: Excess of adherence to the agent technology standards can hinder the software development process while the focus in good practices can lever this process. This paper presents IDEA, a lightweight microservice-oriented platform that facilitates the development and execution of Multi-Agent Systems in the business context. The solution design seeks for a good trade-off between usability and adherence to the agent technology standards. The platform also enables microservices discovering, composition, and reuse.

1 INTRODUCTION

Currently, most of the organizations adopt heterogeneous and complex information systems enabled by low coupled components. These systems must coordinate their components in order to provide efficient support to business processes and consistent information management. Enterprise and business applications running in the Web are increasing in complexity, involving widely heterogeneous entities, many functions, and the interaction among several devices.

In the industrial context, the Service Oriented Architecture (SOA) principles have evolved from the original service concept to a new concept of microservices. Microservices paradigm is based on independent and low-granular software components that interact to build highly scalable systems (Dragoni et al., 2017). Rooted in SOA principles, this new style of architecture has gained strength in recent years, and are disseminated by companies such as Netflix, Spotify, Uber, Amazon, and PayPal (Lewis, 2015).

If Lewis et. al. (2010) already recognized SOA as the best option available for system integration and leverage of legacy systems, the recent advent of microservices increased significantly this importance. Technologies to implement SOA will certainly evolve to address emerging needs, but its basic principles seems to keep stable. One of the primary benefits of SOA is the ability to cdompose applications, processes, and assemble new functionalities from existing services.

There are several authors researching the application of microservices paradigm as a framework for building modern agent-based systems (Cabri et al., 2010; Collier et al., 2015; Crow et al., 2018; Higashino et al., 2018). As argued by the authors, the inability of agent's technology for dealing with complex communication protocols and also with the excessive formalism of agent-oriented methodologies contributes to inhibit their use in the business context.

The combination of microservices-based approaches with intelligent agents can be a way to compose and develop intelligent solutions. Agents can dynamically discover, orchestrate, and compose microservices, check activities, run business processes, and integrate heterogeneous and distributed applications. An important aspect of microservices is to provide conditions for the system to adapt according to requirements or circumstances. Evolution can be applied on time, in features they need (Silva 2017).

This paper presents IDEA, a lightweight microservice-oriented platform for development of intelligent agent-based applications. IDEA aims at offering an infrastruture to facilitate the development and execution of intelligent MAS in an easier and faster fashion. The platform also enables microservices discovering, composition, and reuse.

Haendchen Filho, A., Ribeiro, R., Antônio do Prado, H., Ferneda, E. and Thalheimer, J.
A Lightweight Microservice-oriented Platform for Development of Intelligent Agent-based Enterprise Applications.
DOI: 10.5220/0010311003760383
In *Proceedings of the 13th International Conference on Agents and Artificial Intelligence (ICAART 2021) - Volume 1*, pages 376-383
ISBN: 978-989-758-484-8

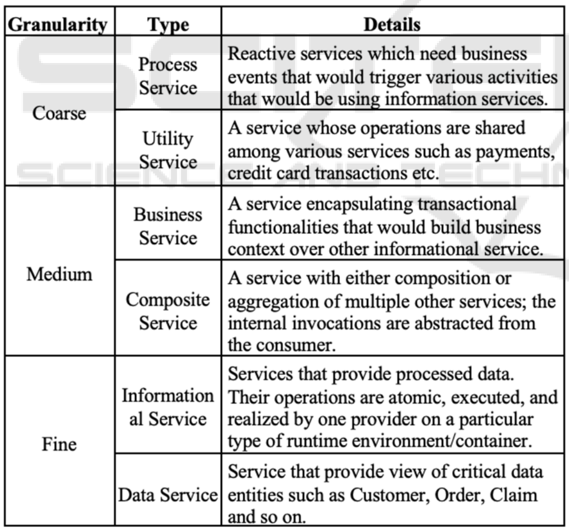
2 BACKGROUND

This section describes relevant aspects related to resources and services, and a well-known agent modeling methodology used in our approach.

2.1 Resources and Services

Large and medium sized organizations usually can have hundreds and even thousands of fine-grained procedures distributed across business applications. Services of the most value to business experts are constructed from lower-level services, components, and objects that are structured to meet specific business needs. Elimination of redundancy assemble of new functionalities from existing services, adaptation of systems to changing needs, and leverage of legacy investments are the common goals for the adoption of SOA (Lewis and Smith 2007).

As different granularity services normally fulfil different roles, coarse, medium, or fine granularity categories of services are considered. These categories can be classified by type, as shown in Figure 1 (adapted from Kulkarni and Dwivedi, 2008).

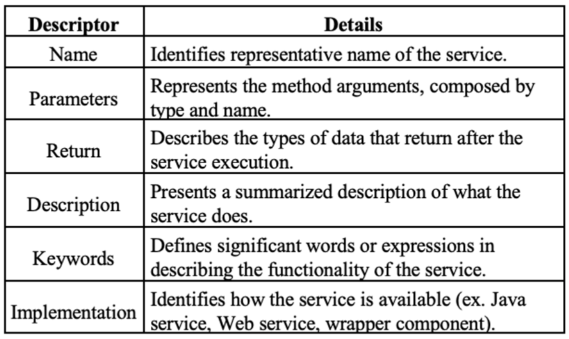


Granularity	Type	Details
Coarse	Process Service	Reactive services which need business events that would trigger various activities that would be using information services.
	Utility Service	A service whose operations are shared among various services such as payments, credit card transactions etc.
Medium	Business Service	A service encapsulating transactional functionalities that would build business context over other informational service.
	Composite Service	A service with either composition or aggregation of multiple other services; the internal invocations are abstracted from the consumer.
Fine	Informational Service	Services that provide processed data. Their operations are atomic, executed, and realized by one provider on a particular type of runtime environment/container.
	Data Service	Service that provide view of critical data entities such as Customer, Order, Claim and so on.

Figure 1: Services classification chart.

SOA and services in general rely strongly on the concept of resource. Human and software agents can discover services and provider entities by means of the metadata used to describe a resource (Bhuvaneswari and Sujatha 2011). The fine- and medium-grained services need to be appropriately described, represented, and exposed in order to be known and visible to modelers, developers, and business analysts involved in the composition of medium and coarse granularity services.

The mechanisms for invocation of services such as descriptors and discovery must comply with established internal standards. Figure 2 shows the services descriptors.



Descriptor	Details
Name	Identifies representative name of the service.
Parameters	Represents the method arguments, composed by type and name.
Return	Describes the types of data that return after the service execution.
Description	Presents a summarized description of what the service does.
Keywords	Defines significant words or expressions in describing the functionality of the service.
Implementation	Identifies how the service is available (ex. Java service, Web service, wrapper component).

Figure 2: List of service descriptors.

The first two descriptors are automatically retrieved by applying Java Class Library methods, and the other are informed by the developer. The Name, Return, Description, Keywords and Implementation descriptors are stored in meta classes.

2.2 The Role Model

The role model originates from Biddle and Thomas (1966), in which a role refers to an expected pattern of behavior for an agent. In the Software Engineering context, role models were applied initially for defining a set of entities behavior in object-oriented approaches (Cabri et al. 2010). After that, the role model was applied for MAS design and development.

The first principle of the behavior-based approach for MAS development is that each entity must have a unique role, clearly defined by a coherent block of functions or services. The importance of using roles is emphasized by the fact that it can be adopted in different areas of computer systems to obtain decoupling at different levels (Cabri et al. 2010).

Ferber and Gutknecht (1998), Fasli (2003), Cabri et al. (2003), and Zambonelli et al. (2001), among others, stablished the basis for applying role models in MAS development. In this work, Gaia Methodology (Zambonelli et al., 2001) was adopted. Gaia's main objective is to model multi-agent systems as organizations where different roles interact. Roles are considered only in the analysis phase.

Gaia defines roles by means of the following attributes: *(i)* Responsibilities, that specify the agents functionalities in order to perform the related roles; *(ii)* Permissions are a set of rights that mainly refer to the agent data access level; *(iii)* Activities are internal computations of agents, which may involve calling services, or functions that do not requires interaction with other agents; and *(iv)* Protocols which specify how an agent performing its role can interact with other agents.

Gaia methodology enables an organizational view of the system. The requirements definition is made during the analysis phase and is the first step in the development.

3 IDEA AGENT PLATFORM

This work presents IDEA, an evolution of MIDAS (Haendchen Filho, 2007), introducing the microservice concept for MAS development. The architecture is based on the coexistence of several containers that communicate by means of a front-end server. Each container provides an environment for development and execution of agents. Figure 3 shows the generic architecture.

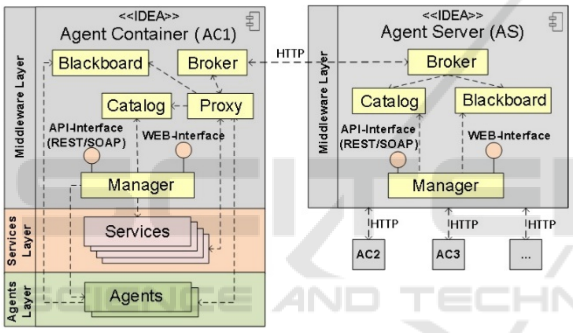


Figure 3: IDEA generic architecture.

The architecture consists of three layers: *(i)* Middleware, represented by the entities Broker, Proxy, Catalog, Manager, and Blackboard; *(ii)* Services; and *(iii)* Agents. The middleware layer facilitates the development by abstracting complex developer procedures. It provides communication, concurrency, lifecycle management, and discovery. The Agents and Services layers enables the development of agent-based applications.

The Agent Container (AC1) represents the containers where applications can be instantiated. The communication between Agent Layer and Service Layer is done by the Proxy, in the middleware layer. For short, the agent does not need to know who is the service provider, a feature that ensures transparency and decoupling from the implementation. Services can be created independently, and agents can use and coordinate these services in their workflow.

Agent Server (AS) is responsible for implementing the platform integration rules, synchronizing the containers, maintaining a catalog of all services, and calling remote services. There is no Proxy in AS because its role of creating agent instances and invoke services is performed only in the containers.

AC and AS have three interfaces: *(i)* HTTP for inter-platform communication between the front-end server AS and the AC containers; *(ii)* an API interface, which enables the communication with external applications via REST/SOAP protocols; and *(iii)* a WEB-Interface for platform management and configuration by developers.

3.1 Middleware Layer

3.1.1 Management Model

Playing the roles defined by the management model, the Manager Agent is the most complex agent in the architecture. It collaborates with the other agents and has its responsibilities distributed along five packages: *manager*, *screens*, *listeners*, *tasks* and *execution*. Figure 4 shows a partial view of its internal structure.

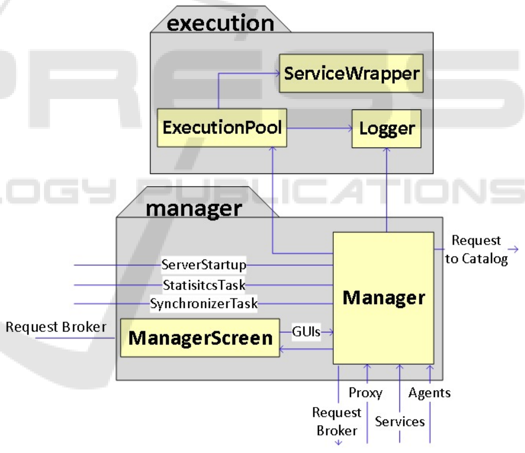


Figure 4: Partial view of the manager model.

The *manager* acts as a gateway receiving/sending request. It is also responsible for the initialization of the network server, platform agents, and applications agents. It collaborates *(i)* with the Catalog agent to create service wrapper objects, *(ii)* with the Broker to synchronize verification requests, and *(iii)* with the Proxy, handling requests for queuing messages. Manager Screen is a management class for accessing the graphical user interfaces.

The purpose of the *execution* is to allow more intuitive and flexible operations to be carried out in the process of building a service request. In addition, it creates an execution pool with queuing and request execution. This package encompasses three classes: *(i)* ServiceWrapper, making very simple the construction of a service request by enabling the specification of parameters without concerns about their sequence, location or type conversion; *(ii)* Logger, which implements a log file that is used to store information about the processing of requests and statistics; and *(iii)* ExecutionPool, controlling the execution pool for synchronous and asynchronous calls.

3.1.2 Communication Model

The communication model defines the way agents communicate and how messages traverse service providers and requesters. Communication can occur in synchronous or asynchronous mode.

Synchronous Communication. Communication among agents in the synchronous mode can occur in three ways: extra-, inter- and intra-platform. Extra-platform messages are performed via REST-SOAP and is received by the Manager class placed in *manager*, as shown Figure 5. It manipulates messages extracting the XML message or parameters into a native protocol. After that, it delegates the request for the current business process (or entity).

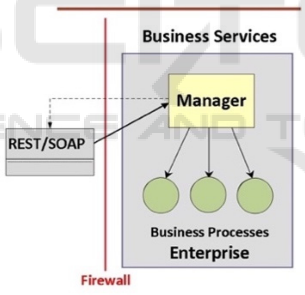


Figure 5: Extra-platform communication.

Inter-platform communication occurs when an agent placed in an AC container requests a service from another agent located in a remote container. The request is sent to the Proxy, that processes and checks whether it the service can be made available locally. Otherwise, it forwards the message to the Broker agent (in the container), that drives it to the AS. In AS, the Broker agent identifies the service provider container and forwards the message. Intra-platform communication are the messages exchanged among agents in the same container. They always are done via Proxy that identifies the provider, creates the instance, and invoke the service.

Asynchronous Communication. Blackboard (Erman et al., 1980) is one of the most used information exchange techniques in symbolic cognitive systems. It acts as sensors that perceive changes in the environment, addressing them to those agents that implement the MessageListener and ContextListener interfaces (Figure 6).

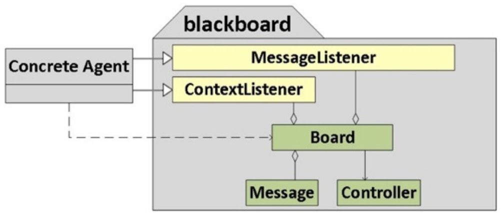


Figure 6: The *blackboard* architecture.

Board class is responsible for listeners registering. When the *blackboard* is initialized, all agents are registered as potential listeners, using their name as an identifier. Agents can register with interest groups, sending messages to everyone, to groups or to an individual agent. During the resolution of a problem, the actions taken by the agents gradually modify the data structure, and the state of the solution. Board also makes a data structure visible for all agents.

The Message class offers a set of services to keep messages coming to the blackboard and its respective log. Agents can access this log file and retrieve data structured by group, date, subject, and so on. Controller class is responsible for monitoring and notifying agents or groups of agents about changes in the blackboard. It acts as a sensor for agents, capturing changes in the data structure and notifying interested agents or groups of agents.

3.1.3 Service Model

Playing the roles defined by the Service Model, the Proxy agent acts as a representative of the service provider. It is responsible for processing service requests. It plays a key role because it acts as a bridge between agents and the presentation and data layers. It reduces the complexity of the code and avoids the implementation of controller classes to mediate the two layers. Figure 7 shows the structure of the Proxy agent, composed by two classes: Proxy and Factory.

Factory class plays an important role in making the architecture flexible and adaptable. It performs procedures of reconfiguration and dynamic creation of instances. Reconfiguration makes possible for a class running on the JVM to be replaced at run time. Thus, new agents can be inserted dynamically into the platform without to change the Factory class code.

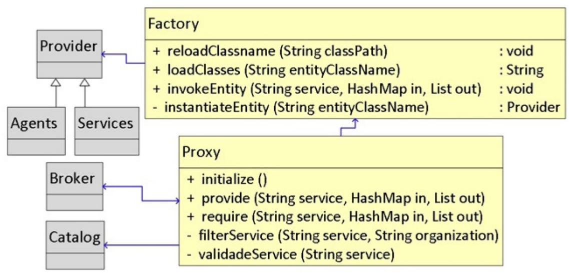

Factory
+ reloadClassname (String classPath) : void
+ loadClasses (String entityClassName) : String
+ invokeEntity (String service, HashMap in, List out) : void
- instantiateEntity (String entityClassName) : Provider
Provider
Agents
Services
Broker
Catalog
Proxy
+ initialize ()
+ provide (String service, HashMap in, List out)
+ require (String service, HashMap in, List out)
- filterService (String service, String organization)
- validadeService (String service)

Figure 7: Service model architecture.

3.1.4 Resource Model

Resource is a fundamental concept that underpins much of the SOA and services. A resource description are metadata that makes possible for a human or software agent to discover service and provider entities (Bhuvaneswari and Sujatha, 2011). Figure 8 shows the internal structure of the Catalog, composed by *catalog* and *metainfo* packages. The Catalog class (in the *catalog*) is a gateway, able to handle service requests from agents via Proxy.

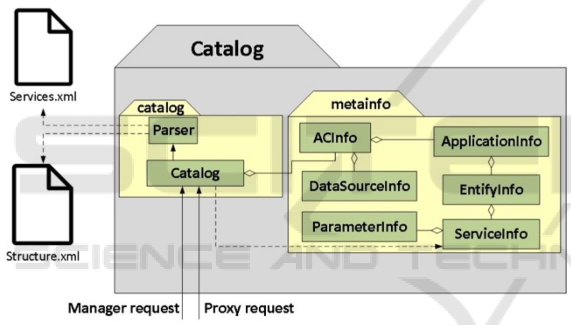


Figure 8: Model architecture of Catalog agent.

In the *metainfo* package, services are organized in a hierarchical parent/child entity set. Each element of the structure is represented by a metaclasse implemented as an entity. It stores specifications of data sources in the DataSourceInfo entity, about the service in the ServiceInfo entity, and so on. The resources are described in the Services.xml file.

3.2 Services Layer

Unlike agents, services are purely reactive entities, which have no thread of execution, although they can execute concurrently. Services may be developed in different granularities, as shown Section 2.1. The abstract class Services can be used as a wrapper, capable of encapsulating data access objects, domain-specific functionality, legacy applications, parts of business processes and other procedures. It is common these services to be implemented for database access or other procedures.

3.3 Agents Layer

In the agent layer, the abstract class Agents defines an algorithm skeleton, including concrete and abstract methods. The concrete methods provide a set of already implemented functions, such as the interfaces for interacting with the architecture, the procedures that start the agent execution thread, and the basic signatures for the agents' lifecycle management. More than simple entry-points, the abstract classes empower the agents with facilities, such event-based listeners, and implements a blackboard interface. Agents offers two interfaces for external communication: *(i)* a provider interface, by which the agent can receive services call or collaboration requests; and *(ii)* a requester interface, by which the agent can request external services.

4 PRACTICAL SAMPLE

In order to validate the IDEA platform, some multi-agent systems have been developed, like VLE-MAS (Thalheimer, 2020), a Virtual Learning Environment (VLE). VLE-MAS works in an abstraction layer above the operational layer of the different platforms for distance learning, such as Moodle. The goal was to use intelligent software agents to make current VLEs more proactive, dynamic, and interactive. Agents monitor the environment, organize data, send warning signals to tutors, teachers, and promote interactivity with students.

Figure 9 presents the VLE-MAS architecture, including: *(i)* the IDEA platform; *(ii)* a data webhouse for representing the data models; *(iii)* VLE Multi-Agent System composed by a set of collaborative agents; and *(iv)* the Moodle virtual environment.

The VLE-MAS is composed by the following collaborative agents: *(i)* Tracing Agent (TAg), that manages the data structure; *(ii)* Interface Agent (IAg), that performs interactions with human actors; *(iii)* Knowledge Agent (KAg), for generating predictions and prescriptions; *(iv)* Pedagogical Agent (PAg), which performs content management, learning objects and trails; and *(v)* Student, Tutor and Professor, represent virtual instances of these human actors. TAg is presented in this sample.

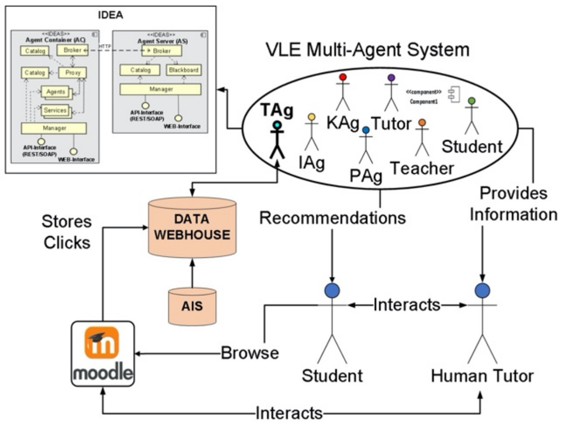


Figure 9: Generic architecture of the VLE-MAS system.

Gaia methodology, adapted to the context of services, was applied for analysis and design. Figure 10 presents the role model of TAg agent. Tag's responsibilities are divided into three groups: *(i)* Extracting and Monitoring; *(ii)* Storing; and *(iii)* Interacting. It is possible to observe the differences in granularity of services. Each group represents a coarser granularity or composite service, built over a set of finer granularities microservices. Permissions, collaborations and interactions, in the right-side column, state the data the agent is allowed to access.

ROLE: Tracing Agent ORGANIZATION: VLE Multiagent System (VLE-MAS)	
RESPONSIBILITIES	**PERMISSIONS**
Extracting and Monitoring	Student profile data Student school historic Teachers data Real time feedback
Extracting data from academic Information systems	
Extracting data from Moodle log file	
Cleaning and transforming collected data	
Monitoring the triggers from database	**COLLABORATIONS**
Storing	Pedagogical Agent Knowledge Agent
Populating the DW dimensions	
Populating the Fact table	**INTERACTIONS**
Interacting	IAG through BB IAg and PAg
Notifying agents about triggers	
Notifying agents about interest relevant events	

Figure 10: TAg's role model chart.

The TAg's responsibilities are modeled with the HEFLO tool (https://app.heflo.com). It has strong adherence to the Business Process Model and Notation (BPMN). It is intuitive and allows to represent complex process details in an understandable language in the business context.

Figure 11 shows a workflow of the Notifying Triggers service, as well as collaboration and interaction among agents and entities. The workflow starts when the student logs in the Moodle. After logged, the Proxy creates a student instance and sends its ID to the TAg. Next, TAg receives a trigger message from DBPool and writes this message in the Blackboard. The IAg is notified, reads the message and performs an interactive activity with the tutor, completing the flow. DBPool is an entity of IDEA responsible for managing access to databases.

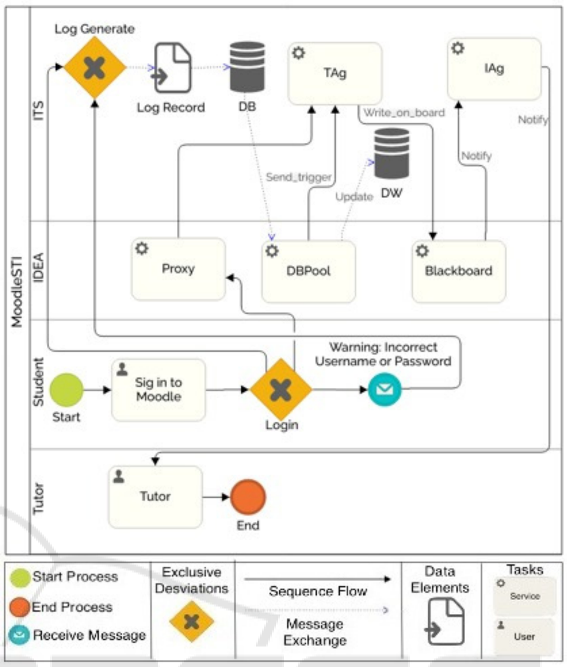
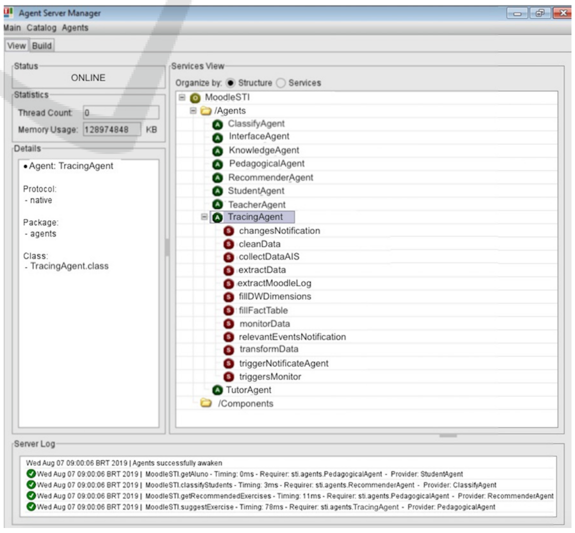


Figure 11: Notifying Triggers service workflow.

Figure 12 presents the GUI wizard for VLE-MAS application, including agents and the services implemented by TAg.



```
Wed Aug 07 09:00:06 BRT 2019 | Agents successfully awaken
Wed Aug 07 09:00:06 BRT 2019 | MoodleSTI.getAluno - Timing: 0ms - Requirer: sti.agents.PedagogicalAgent - Provider: StudentAgent
Wed Aug 07 09:00:06 BRT 2019 | MoodleSTI.classifyStudents - Timing: 3ms - Requirer: sti.agents.RecommenderAgent - Provider: ClassifyAgent
Wed Aug 07 09:00:06 BRT 2019 | MoodleSTI.getRecommendedExercises - Timing: 11ms - Requirer: sti.agents.PedagogicalAgent - Provider: RecommenderAgent
Wed Aug 07 09:00:06 BRT 2019 | MoodleSTI.suggestExercise - Timing: 78ms - Requirer: sti.agents.TracingAgent - Provider: PedagogicalAgent
```

Figure 12: GUI wizard displaying the TAg services.

For services reusing, clients need to know much more than a simple service name or the address of the service provider. Developers perceives a service as an interface, including their necessary parameters. GUI wizard provides filters for selecting services by attributes like name, function, and keywords, facilitating its location. In the left side, specifications of the selected service are displayed.

5 RELATED WORKS

JADE (Bellifemine et al. 2001) is a platform that aims to facilitate MAS development. The architecture is based on the coexistence of several Java virtual machines, having the communication between different virtual machines done via Java RMI. Each virtual machine is a basic agent container, which provides an environment for running agents and allows multiple agents to run concurrently on the same host. Even though it was created almost twenty years ago, JADE remains a reference. Its architecture follows the reference model FIPA (2000), being composed by three main agents in the infrastructure layer: *(i)* Management System Agent, that performs supervision and control over the access and use of the platform; *(ii)* Communication Channel Agent, the connection among agents inside/outside the platform; and *(iii)* Directory Facilitator, that provides the yellow pages service, which can be accessed by agents and viewed by developers.

WADE (Banzi et al. 2008; Banzi et al. 2017) is a software platform based on JADE that provides support for the execution of tasks defined according the workflow metaphor. WADE is a trademark of the Telecom Italia SpA. Its key component is the WorkflowEngineAgent class that extends the basic Agent class of the JADE library embedding a small and lightweight workflow engine. Besides normal behaviors, a WorkflowEngineAgent is able to execute workflows represented according to a WADE specific formalism. The approach enables to combine the expressiveness of the workflow metaphor with programming languages like Java.

JaCaMo (Boissier et al., 2020) is a framework for Multi-Agent Programming that combines three well-known technologies: *(i)* Jason, for autonomous agents programming; *(ii)* Cartago, for environment artifacts programming; and *(iii)* Moise, for multiagent organizations programming. A MAS is designed and programmed as a set of agents which work and cooperate inside a common environment. Agents operating in this context follow the organizational constraints defined by means of artifacts which provide the agents' functionalities and the operations giving access to these functionalities.

Python Agent DEvelopment (PADE) (Melo et al., 2019), is an open-source platform implemented in Python and conceived for MAS implementation on power systems. It is compliant with specifications of FIPA and eases the development of solutions to power systems based on MAS. By means of PADE, developers are building smarter power grid systems. Smart grids employ digital technology and are based on highly collaborative and responsive decision-making strategies.

6 DISCUSSION AND CONCLUSIONS

IDEA offers several facilities and some important advantages in comparison with the presented approaches. First, none of the approaches are service oriented, making difficult to adapt and reuse for different levels of granularity. Second, it is more difficult to design a MAS having to tackle with composition and reuse details, instead of designing only to meet a specific function.

For each agent service, JADE requires the creation of a specific behavior intraclass that describes programmatically the interface. A new service request involves the handling of a controller class, which needs to be programmed to manage the communication among the application layers. Depending on the system size, this class tends to become extremely complex and liable to errors.

For both JADE and WADE, the specification of a FIPA-ACL message structure specification is not trivial. It involves a set of tags for different communicative acts, expressions, and so on. IDEA provides simplicity to the process of creating, sending and receiving messages. It requires a much simpler level of coordination for handling requests than the approaches associated with the concept of messages and theirs protocols.

The availability of a workflow engine in WADE, as a JADE extension, makes the solution able to perform workflows representing behaviors. However, the already pointed limiting characteristics in the FIPA core remain and can inhibit the large-scale use in the context of business application.

Although its ability for enabling the development under four layers of abstraction, JaCaMo is not easily understandable. It presents a high level of difficulty that does not match with the simplicity required for developing large scale MAS. A particular difficulty is understanding its conceptual model.

PADE is tailored to work with advanced power grids. So, it falls out of the enterprise and business application context, central for the ideas presented here.

Beyond the comparative advantages provided by IDEA, it is important to notice that none of them mention the ability to dynamically configure agents and services. Intuitively, the asynchronous communication model via blackboard used by IDEA may be more efficient than those used in the analyzed approaches. Handling asynchronous messages via blackboard enables generating message logs, creating a knowledge base that can be very useful for applying machine learning techniques. In addition, it facilitates the sending of broadcast messages, messages to group of agents, or messages to a specific agent.

The main contribution of this work is a microservice-oriented platform to facilitate and speed up the MAS development in the business context. The solution design seeks for a good trade-off between usability and adherence to the standards. Excess of adherence to standards can hinder the development while to focus in good practices can lever this process. As future works, the creation of a layer for IOT handling and make IDEA an open-source project.

REFERENCES

Banzi, M., Caire, G., Gotta, D., Pota, M. 2017. WADE - Workflows and Agents Development Environment. https://jade.tilab.com/wadeproject/.

Banzi, M., Caire, G., Gotta, D., 2008. WADE: A software platform to develop mission critical applications exploiting Agents and Workflows. In *AAMAS 2008, 7th International Joint Conference on Autonomous Agents and Multiagent Systems: Industrial Track*.

Bellifemine, F., Poggi, A., Rimassa G., 2001. JADE: a FIPA compliant agent development environment. In *AGENTS '01, Fifth international Conference on Autonomous Agents*. ACM Press.

Boissier, O., Bordini, R. H, Hübner, J., Ricci A., 2020. *Multi-Agent Oriented Programming: programming MAS using JaCaMo*, MIT Press.

Bouman, R., Dongen, J. V., 2009. *Pentaho Solutions: Business Intelligence and Data Warehousing with Pentaho and MySQL*, Wiley Publishing.

Bhuvaneswari, N. S., Sujatha, S., 2011 *Integrating SOA and Web Services*, River Publishers.

Cabri G., Leonardi L., Ferrari L., Zambonelli F., 2010. Role-based software agent interaction models: a survey. *The Knowledge Engineering Review*, 25(4):397-419.

Cabri G., Leonardi L., Zambonelli F., 2003. BRAIN: a framework for flexible role-based interactions in multiagent systems. In *CoopIS*, *2003 Conference on Cooperative Information Systems*.

Collier, R. W., Russell, S. E., Lillis, D., 2015. Reflecting on Agent Programming with AgentSpeak. In *PRIMA 2015, 18th International Conference Principles and Practice of Multi-Agent Systems*.

Crow, T., Luxton-Reilly, A., Wuensche, B., 2018. Intelligent tutoring systems for programming education. In *ACE '18*, *20th Australasian Computing Education Conference*, p. 53-62.

Dragoni, N., Giallorenzo, S., Lluch, A., Montes, F., Mustafin, R., Safina, L. 2017. Microservices: Yesterday, today, and tomorrow. In *Present and Ulterior Software Engineering*. Springer. p. 195–216.

Erman, L. D., Hayes-Roth, F., Lesser, V. R., Reddy, D. R., 1980. The Hearsay-II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty. *ACM Computing Survey* 12:213-253.

FIPA, 2000. *FIPA Reference FIPA-OS V2.1.0*, Nortel Networks Corporation.

Haendchen Filho, A., Prado, H. P., Lucena, C.J.P. 2007. A WSA-based architecture for building multiagent systems. Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems, AAMAS 2007.

Higashino, M., Kawato, T., Kawamura, T., 2018. A Design with Mobile Agent Architecture for Refactoring A Monolithic Service into Microservices. *Journal of Computers*, 13(10).

Kulkarni, N., Dwivedi, V. 2008. The role of service granularity in a successful SOA realization – A Case Study. In *2008 IEEE Congress on Services*.

Lewis, G. A., Smith, D. B., 2007. Four pillars of Service-Oriented Architecture. The Journal of Defense Software Engineering, Carnegie Mellon University, Software Engineering Institute.

Lewis, G. A., Smith, D. B., Kontogiannis, K., 2010. A Research Agenda for Service-Oriented Architecture: Maintenance and Evolution of Service-Oriented Systems. Technical Note, CMU/SEI-2010-TN-003.

Melo, L. S., Sampaio, R. F., Leão, R. P. S., Barroso, G. D., Bezerra, J. R., 2019. Python ‐ based multi ‐ agent platform for application on power grids. *International Transactions on Electrical Energy Systems*, 29(6).

Silva, B. D, 2017. Dossiê: Modelo de Confiança para Sistemas Multiagentes. PhD Dissertation – Pontifícia Universidade Católica do Paraná, v. 1, 2017. 134 p.

Thalheimer, J. M., 2020. A tracing agent in the context of collaborative agents for monitoring and managing the data structure in virtual learning environments. MSc Thesis. UNIVALI. (in Portuguese).

Zambonelli, F., Jennings, N. R., Wooldridge, M., 2001. Organisational Abstractions for the Analysis and Design of Multi-agent Systems. In P. Ciancarini, M.J. Wooldridge, *Agent-Oriented Software Engineering*, LNCS, 1957:235-251, Springer-Verlag.