A System of Systems Architecture for Supporting Decision-Making

Gregory Moro Puppi Wanderley, Marie-Hélène Abel, Jean-Paul Barthès, Emerson Cabrera Paraiso

A System of Systems Architecture for Supporting Decision-Making

Gregory Moro Puppi Wanderley, Marie-Hélène Abel, Jean-Paul Barthès
Sorbonne Universités, Université de Technologie de Compiègne
CNRS UMR 7253 Heudiasyc
57 Avenue de Landshut - CS 60 319 - 60 203 Compiègne Cedex, France
Email: {gregory.wanderley, marie-helene.abel, barthes}@utc.fr

Emerson Cabrera Paraiso
Pontifícia Universidade Católica do Paraná
PPGIa - Graduate Program in Informatics
1155 I. Conceiçao - 80215-901 Curitiba, Brazil
Email: paraiso@ppgia.pucpr.br

***Abstract*—Good Decision Support Systems require three main features: (i) a good handling of the domain data and information; (ii) an efficient user interface; and (iii) a good knowledge of past decisions. Usually such features are handled by different specialized systems difficult to integrate. In this research we keep specialized systems independent, focusing on interoperability. We propose a *system of systems architecture* (SoS) integrating a domain system in which users interact, a multi-agent system implementing an efficient user interface and taking into account results from the domain system, and a platform to capitalize and manage knowledge. Our approach extracts indicators from the interaction or behavior of users within the domain system, and provides them analyses, statistics and recommendations to help them reach good decisions. We built a prototype and applied it to two different domains: collaborative software development and healthcare. In this paper, we will focus on the multi-agent system which is a key component of the SoS architecture.**

I. Introduction

Decision Support Systems (DSS) are specific computer-based applications that help making decisions based on the information available (Suni and Gopakumar [1]). Many of the decision-making processes such as team coordination, traffic control, business strategy planning, resolving conflicts or game playing are complex, unpredictable and hardly observable. Moreover, the operators, decision makers, managers or game players have to face the problem of controlling such systems under uncertainty or the existence of random factors (Kwasigroch and Grochowski [2]).

When working in a particular domain, say developing software, decisions are related to selecting algorithms, variables, program structures, syntax and many other things. Decisions result from the know-how and experience of the developer. Developers get some help from IDEs that are tools to facilitate the writing of the code. Because such tools are generic they are limited regarding the amount of help they can provide. While writing the code it would help to have some idea of its quality, benefit from expert advice, be reminded of previous projects, reuse historical data and follow good practices. Other environments and tools exist for providing the information, but they are not integrated and it is difficult to use all of them simultaneously.

Our research consists in designing an environment combining a platform taking care of the targeted application (e.g. a development environment), a platform for adding services, and a platform for managing knowledge. Because such platforms already exist, our goal is to render them interoperable by proposing a System of Systems (SoS) architecture in which each component is itself an independent complex operational system, interacting to achieve a common goal (Saleh and Abel [3]) and loosely coupled to the other ones. The easiest approach to interoperability using loose coupling is the multi-agent approach in which each platform can be extended to handle external connections.

Multi-agents (MAS) have already been used to develop Decision Support Systems (Othman et al. [4], Ying et al. [5], Ling et al. [6]). Software agents have been found to be useful for providing intelligent problem-solving mechanisms and for improving the decision-making processes and therefore obtaining a more powerful decision support (Ling et al. [6]). Moreover, agents can also learn the behavior of users and provide a customized support.

The main contribution of this paper is the proposed SoS architecture that supports interoperability between the involved systems and lets it be applied to different domains more easily. Furthermore, this architecture allows teams to perform their works in a distributed and asynchronous way. Thanks to its characteristics, the approach can give customized support to team members in order to make good decisions.

In this paper we focus on the multi-agent aspect of the SoS, detailing our approach on the case of collaborative software development. We then present a prototype, ending with conclusions and future work.

II. Multi-agent Systems

Multi-agent systems are sets of agents that interact to coordinate their behavior often to achieve challenging tasks (Ren and Chen [7]). Three types of agents are interesting for our purposes: Service Agents, Personal Assistants and Transfer Agents.

- Service agents (SA) provide specific services tailored to the current application. Since they are loosely coupled, they can be added as needed to increase the power of the analysis of data;
- Personal Assistants (PA) act autonomously and are built to be real assistants or surrogates of their master (Wanderley et al. [8]), or even act as “digital butlers” (Negroponte [9]). A PA has a crucial function, being dedicated to:

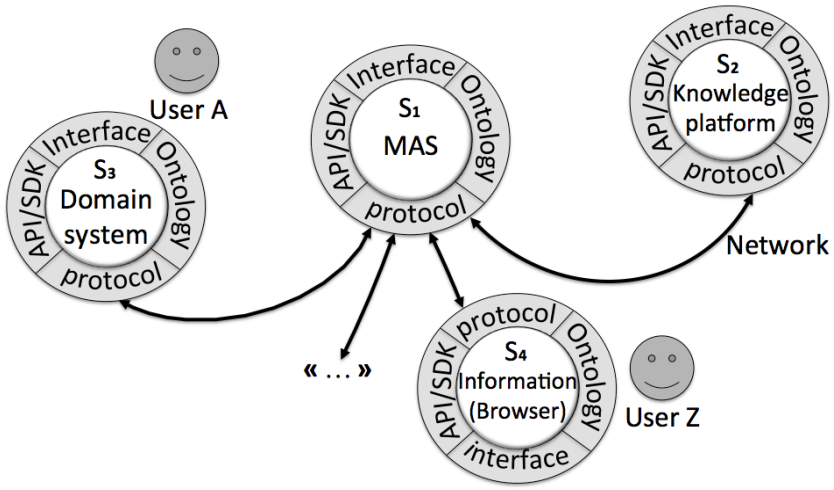

Fig. 1. The proposed Architecture.

- understanding its master's needs (i.e. the needs of the user owning the agent);
- acting pro-actively to anticipate its master's needs;
- mobilizing SAs to execute a command or demand from its master;
- mediating all information exchanges among team members (who are considered information sources);
- organizing the documentation of its master with the help of an SA;
- capturing and representing the team members' operations, helping them in the process of preserving and creating knowledge (Paraiso and Barthès [10]).

- Transfer agents (XA) are gateways that communicate with platforms having different structures, implementing their own protocol, and translating communication and content languages;

III. A System of Systems Architecture for Supporting Decision-making

In this section we present the proposed approach. First, we introduce its architecture, and then the process to provide support to users in their decision-making activities.

A. Architecture

Figure 1 shows the proposed architecture. As mentioned earlier, the architecture forms a system of systems composed of three main parts: a given domain system, a multi-agent system (MAS) and a platform to capitalize and manage knowledge. Each system is encapsulated (loose coupling) by an agent. In order to do that, each system to be integrated must have at least an API or SDK.

There are two groups of users: primary users of the domain system, and users wanting to be aware of data or information. The former ones use the domain system to develop their activities, like employees, operators, etc., and the latter are interested in information obtained from the domain system and from the primary users. They could be managers. They can use an interface like a browser to visualize the data.

The knowledge platform has an ontology that models a given domain and aims at storing and documenting recommendations, indicators, analysis, and statistics extracted from the interaction or behavior of users with the domain system. Moreover, it allows users to contribute with and improve stored information, for instance, recommendations, making annotations, comments, descriptions, recommendations; adding resources; or sharing ideas. This way, it contributes to reusing previous knowledge.

Our approach extracts indicators from the interaction or behavior of users with the domain system, and provides them with analysis, statistics and recommendations to aid decision-making.

In the proposed approach, each user has its own Personal Assistant Agent (PA), and the communication uses natural language. It is important to highlight that each agent (PA, SA and XA) has its own ontology (Figure 1), used for understanding the expressions of the content language (*e.g.* messages between agents and systems), to interpret the users' utterances (*e.g.* dialogues between users and their PAs), and to work as a knowledge base. In the latter case, a PA ontology can be used to store in a *User Model* personal information (name, address, email, etc.) of its master, as well as information concerning the interaction or behavior of users and the domain system. The agent can be aware of the experiences, skills and competencies of users, and then give them personalized support. It is thus possible to better understand the users' needs.

B. Supporting primary users

Let us consider a primary user u that belongs to the set of primary users U. During his interaction with the domain system, a set I_V of new values for a given set of indicators I describing the behavior of users are extracted. Formally, the obtaining of the set of new values I_V can be represented as a unary predicate $I_V = \forall i \ I_V(iv) \rightarrow (i, iv) \in I \, X \, I_V$, where i is an indicator belonging to I and iv ($iv \in I_V$) is the new value obtained for i.

After the extraction of the values of indicators, the next step is to analyze and evaluate them being able to identify the ones that the user needs support. This is done by a specific Service Agent (SA) that compares the value *iv* of some indicator with a given set of thresholds (range risks). Formally, the set Z_V of indicators belonging to the range risks can be represented as $Z_{\mathrm{V}} = \forall i, iv \;\; Z_{\mathrm{V}}(i) \rightarrow ((iv \leq D_{\mathrm{l}}(i)) \vee (iv \geq D_{\mathrm{u}}(i)))$, where:

- *i* is an indicator belonging to *I*.
- *iv* is a value obtained for the indicator *i*.
- D_l is a function $- \;\; D_{\mathrm{l}} : I \rightarrow L_{\mathrm{t}} \; -$ that returns the lower threshold l_t ($l_{\mathrm{t}} \in L_{\mathrm{t}}$, set of lower thresholds for the indicators of *I*).
- D_u is a function $- \;\; D_{\mathrm{u}} : I \rightarrow U_{\mathrm{t}} \; -$ that returns the upper threshold u_t ($u_{\mathrm{t}} \in U_{\mathrm{t}}$, set of upper thresholds for the indicators of *I*).

If the SA finds that there are indicators in the range risk, *i.e.* $Z_{\mathrm{V}} \neq \emptyset$, then it will fetch recommendations from the knowledge platform. Formally, let

- *R* be the finite set of recommendations available in the knowledge platform, *r* be a recommendation belonging to *R*.
- *i* an indicator belonging to *I*.

The subset $R^{'}$ ($R^{'} \subseteq R$) of the recommendations found for the indicators in the range risk can be defined by the predicate: $R^{'} \; : \forall i, r \;\; R^{'}(r) \rightarrow (i, r) \in I \, X \, R$. The recommendations found are then sent to the user, concerning the extracted indicators, by his PA.

C. Supporting awareness

The indicators extracted from the interaction of primary users with the domain system are capitalized and stored in the knowledge platform. In order to let users be aware of information about the indicators, an SA retrieves from the knowledge platform charts showing information, analysis and statistics regarding the behavior of primary users with the domain system.

Given the name of some primary user *u* ($u \in U$) or the name of indicators belonging to *I*, the PA of a given user interested in information regarding primary users, will retrieve a chart *g* showing the requested data from the knowledge platform.

Formally, let *g* be a chart belonging to the set of charts *G* that shows analysis and information concerning the value of the indicators stored in the knowledge platform. A single chart *g* is composed of a given primary user, and the indicators obtained from his behavior, *i.e. g(u, C_I)*. The function $- \;\; D_{\mathrm{G}} : U \, X \, I \rightarrow G \; -$ returns the chart *g* that shows the indicators concerning the user *u*.

IV. Case Studies

In this section, we present two case studies from different domains in which we applied the proposed architecture. First, we show our approach in the collaborative software development domain. Then, in the health care domain.

TABLE I. Excerpt of software quality metrics (adapted from [11]).

Metrics	
McCabe's Cyclomatic complexity metric - MCC [12]	Weighted Methods per Class metric - WMC [13]
Lack of Cohesion in Method metric - LCOM* [13], [14]	Nested Block Depth - NBD [15]
Depth of Inheritance Tree - DIT [13]	Number of Children - NOC [13]
Number of Overridden Methods - NORM [14]	Specialization Index - SIX [14]
Method Lines of Code - MLOC [14]	Number of Attributes per Class - NOA [14]
Number of Static Attribute - NSF [16]	Number of Static Methods - NSM [16]
Number of Parameter - NOP [16]	Number of Interfaces - NOI [16]
Number of Package - NOP [16]	Afferent Coupling - Ca [15]
Efferent Coupling - Ce [15]	Instability metric - I, named here as RMI [15]
Abstractness - A, named here as RMA [15]	Normalize Distance from Main Sequence - D [15]

A. Supporting software development teams to improve code quality

In order to support decision-making in collaborative software development, we applied our architecture intending to improve the quality of the code produced by developers. Following, we explain the approach based on what we proposed in Wanderley et al. [11].

The architecture forms a system of systems composed of three main parts: a Software Development (SD) environment, that is, the domain system; an MAS; and a platform to capitalize and manage knowledge. The primary users of the domain system are developers who write their codes in the SD environment. In addition, users interested in information obtained from the interaction and behavior of developers, are managers of collaborative SD.

One of the ways that PAs can support developers is by measuring the code quality and providing recommendations to improve the code quality. During collaborative software development, while a developer is developing code, software quality metrics are calculated and extracted from the code each time the code is compiled. Table I shows some software quality metrics related to complexity; inheritance; size; and coupling, for object-oriented source code.

After the extraction, the resulting metrics are sent to the MAS, through a transfer agent XA that bridges the two platforms. After that, the service agent "Metrics Agent" receives, analyzes and evaluates the metrics, verifying whether the code needs quality improvement. To evaluate the metrics, the agent compares them with thresholds or range risks. If the evaluation concludes that the code needs some improvement, them the Metrics Agent forwards it to the a SA "Search," that will try to find recommendations in the knowledge platform.

Managers on the other hand, have an interface named "Manager - Information," with which they can interact with the environment. Managers may be aware of the quality of projects or the quality of the code developed by a given developer. They receive charts or tables showing the code quality information, aiding him in decision-making.

B. Improving drug prescriptions comprehension and execution

In this case study the proposed architecture was applied to a regional project in France, named CONSIGNELA. The

goal is to support elderly people and patients suffering from Parkinson disease to improve their adherence to a medication regimen prescribed by a caregiver. In this context, primary users of the domain system are the elderly people and the patients with Parkinson. On the other hand, caregivers are the people who want to be aware of information obtained from interaction or behavior of patients and elderly people.

In this case, the system of systems architecture integrates a virtual pillbox, an MAS and a knowledge platform to capitalize and manage knowledge. The virtual pillbox is the tool that patients and caregivers use to follow or prescribe a medication regimen. The MAS provides agents that extract and analyze automatically and in a non-intrusive way, actions and information resulting from the interactions between patients and virtual pillboxes. The information is capitalized and stored in the knowledge platform. Personal Assistant agents (PA) present the information, using charts and dashboards, to caregivers, making them aware whether patients are following correctly the medication regimen, or reporting for example how long a patient spent to take a specific medicine. Moreover, PAs are able to give recommendations to patients to improve their adherence to the medication regimen.

V. Prototype

This section briefly presents a prototype of the proposed architecture concerning the case study of supporting software development teams to improve code quality described above. Currently, the prototype regarding the case study to improve drug prescriptions comprehension and execution is under development.

The prototype is based on our ACE4SD system (Wanderley et al. [11]). It shows the point of view of the developers and managers. In order to build the prototype, the following tools were used: the Eclipse IDE[1], the OMAS platform (Barthès [17]) and the MEMORAe platform (Abel [18]). The Eclipse IDE is a Java environment, in which developers write their codes. The OMAS platform provides the multi-agent environment, and the MEMORAe platform the environment to capitalize and share knowledge.

The Eclipse IDE contains two plugins: (i) Metrics [19]: and (ii) ACE4SD.

- The Metrics plugin calculates, automatically and in a non-intrusive way, quality metrics of Java source code, providing the measured values, the mean and standard deviation for resources (project, class and method). Some of the quality metrics that the plugin calculates are available in Table I. The goal is to extract the quality metrics of the source code that is being produced by a developer and to send it to the ACE4SD plugin.
- The ACE4SD plugin creates the interface between developers and their PAs (which are inside the OMAS platform). In this way, the developers can interact with their PA directly from Eclipse, avoiding switching between different windows from different tools. Moreover, the ACE4SD plugin receives the calculated metrics from the Metrics plugin and sends it a the "Metrics Agent" service agent. This agent is responsible for analyzing the metrics.

[1]More information in: https://eclipse.org/home/index.php

Figure 2 shows a developer's interface of the Eclipse IDE. The interface provides a communication channel between developers and their PA. In the interface, a developer can communicate and exchange information with his PA in natural language directly, avoiding changing windows or tool. Besides, the PA is proactive and it is always monitoring its master. In the example of Figure 2, the PA alerts the developer that a new report with information about problems related to code quality and possible solutions is available. When the developer opens the report, a new window (inside Eclipse) will show its content.

A manager's interface of ACE4SD is shown in Figure 3. The interface enables a manager to be aware of the quality of the projects and the developers. She can see the quality of the whole project, the resources (.java) with the worst quality, and what are their main defects, such as inheritance, coupling, complexity, etc. If the quality of some project is under a defined threshold, it is highlighted in red. Besides, in her interface, the manager is able to interact and exchange information in natural language with her PA.

For instance, in Figure 3 the manager asks more details about a specific developer, and the PA brings her the results showed in Figure 4. In this figure, the manager has a quality chart of the developer, being aware of the quality of the code he produced as well as its problems, such as it is difficult to test, *i.e.* the code contains a high number of decisions paths, and more tests are needed.

VI. Related Work

The work of Botti et al. [20] presents an MAS decision support tool for simulating water-right markets. The goal is to use the approach to assist policy makers in decision processes. The tool is based on a multitier architecture, composed of a presentation (GUI), application processing (software agents) and data persistence (DB). However, as opposed to SoS architectures, multitier architectures are client-server architectures usually running the same application, *i.e.* presentation is the client, and the application processing is the server. Besides, all the system needs to be built from scratch, as opposed to SoS architectures, which integrate existing systems.

Bokhari and Ahmad [21] show a multi-agent architecture for providing decision-support in the context of distance learning on the Web. The system monitors the learning activity of the student and helps him/her throughout the learning process. The architecture is based on four layers, namely human level (interfaces for students and teachers), web level (web portal), system level (agents) and storage level (databases).

Ellouzi et al. [22] propose an architecture for visual clinical decision support for the fight against nosocomial infections. The approach integrates visualization techniques in the KDD (Knowledge Discovery in Databases) steps for the decision-making. The architecture is based on 3 layers, namely interface (interface with humans), data (database) and model (data mining). Each layer has a set of agents to perform the tasks.

The work of Robbins et al. [23] presents an information architecture for a clinical decision support system. The archi-

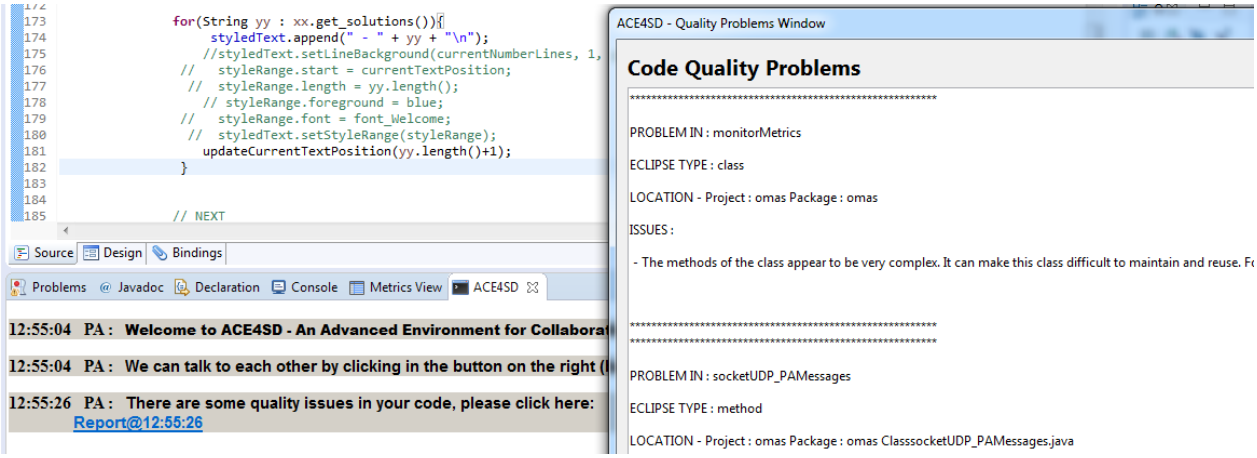

Fig. 2. The ACE4SD developers' interface inside the Eclipse IDE.

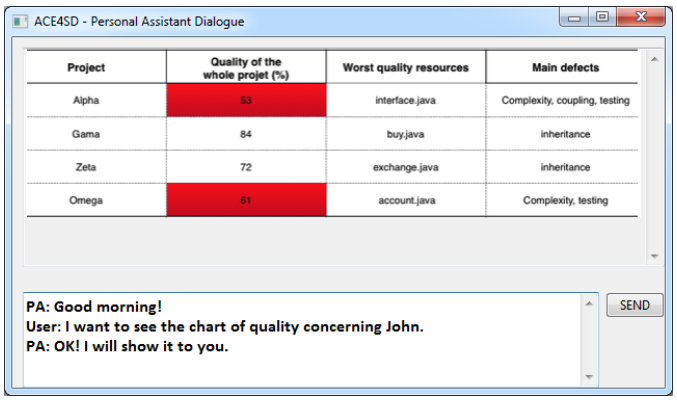

Fig. 3. The ACE4SD managers' interface (adapted from [11]).

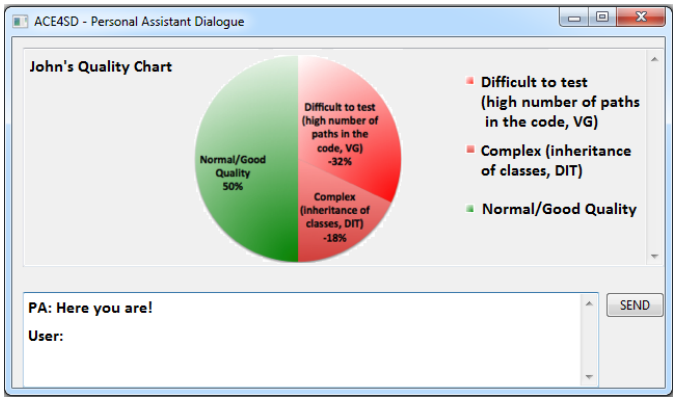

Fig. 4. Details and informations of some developer in the manager's view (adapted from [11]).

tecture comprises a patient model, a treatment library, software agents, and a knowledge base. The patient model contains patient information; the treatment library stores information such as treatment procedures and dosages; agents are used to optimize courses of treatments to best achieve desired health goals. The knowledge base contains medical research used to keep the patient model and treatment library up to date.

An approach to accessing information stored in legacy relational databases, based on Semantic Web (SW) and MAS is also proposed by Polajnar et al. [24]. Their goal is to give the users of enterprise decision-support systems with access to information, through semantic queries, without the need to modify the underlying legacy databases. The authors show an architecture based on two components, namely Database and User. The former comprises a relational database and the server side of the agent-based middleware, whereas the latter provides an interface to the user, and the client side of the agent-based middleware.

The fact is that we could not find an approach to support decision-making based on an SoS architecture integrating a domain system, an MAS and a knowledge platform, all of them loosely coupled. An SoS architecture with loose coupling allows and enhances interoperability. The MAS provides facilities and makes it easier to extend and interface different systems belonging to distinct contexts.

VII. Conclusions and Future Work

Decision Support Systems (DSS) are computer-based applications that help to make decisions based on the information available in different domains. A good DSS involves a domain platform (environment), a set of analysis tools, and a platform for saving and reusing knowledge.

In this research we proposed a system of systems architecture including a given domain system, an MAS and a platform to capitalize and manage knowledge. Each platform is loosely coupled with the others, rendering them interoperable and easier to be applied to different domains. Our approach extracts indicators from the interaction or behavior of users with the domain system, and provides them with analyses, statistics and recommendations to aid decision-making.

In the paper, we presented our architecture focusing on the multi-agent aspects, discussed how we provide support to primary users of a given domain system, and also to users interested in information regarding the interactions and behaviors of primary users and the domain system. Agents are responsible to encapsulate and interface the systems, thus letting them be used in a distributed and asynchronous way. Furthermore, as they have user models of their masters, they are able to provide customized support.

Besides, we applied the proposed architecture to two case studies from two different domains: collaborative software development and health care. In the former, the domain system was a software development environment, and in the latter it was a virtual pillbox.

In addition, we described a prototype regarding the case study of collaborative software development. In the prototype the software development environment was handled by the Eclipse IDE, the multi-agent platform OMAS, and the knowledge platform MEMORAe. We showed the point of view of developers and managers.

Even though our architecture allows and enhances interoperability, it has some requirements. For instance, in order to extend or interface different domain systems, it is necessary to use a common communication protocol between the domain system, the MAS platform and the knowledge management system.

We are currently adapting the prototype to the domain of health care. We plan to test our architecture in other domains.

VIII. Acknowledgment

Gregory Moro Puppi Wanderley would like to thank CNPq-Brazil (process 233137/2014-9) for its support in this research.

Moreover, the authors are thankful to the support provided by the CONSIGNELA project, which is funded by the Regional Council Hauts-de-France and the European Regional Development Fund (FEDER).

References

[1] S. Suni and K. Gopakumar, "A real time decision support system using head nod and shake," in *Circuit, Power and Computing Technologies (ICCPCT), 2016 International Conference on*. IEEE, 2016, pp. 1–5.

[2] A. Kwasigroch and M. Grochowski, "Evolving neural network as a decision support systemcontroller for a game of 2048 case study," in *Methods and Models in Automation and Robotics (MMAR), 2016 21st International Conference on*. IEEE, 2016, pp. 549–554.

[3] M. Saleh and M.-H. Abel, "Information systems: Towards a system of information systems," in *KMIS 2015 7th International Conference on Knowledge Management and Information Sharing*, 2015, pp. 193–200.

[4] S. B. Othman, H. Zgaya, S. Hammadi, A. Quilliot, A. Martinot, and J.-M. Renard, "Agents endowed with uncertainty management behaviors to solve a multiskill healthcare task scheduling," *Journal of Biomedical Informatics*, vol. 64, pp. 25–43, 2016.

[5] Y. Shen, J. Colloc, A. Jacquet-Andrieu, and K. Lei, "Emerging medical informatics with case-based reasoning for aiding clinical decision in multi-agent system," *Journal of biomedical informatics*, vol. 56, pp. 307–317, 2015.

[6] L. Xue, Y. Zhu, and Y. Xue, "Raedss: An integrated decision support system for regional agricultural economy in china," *Mathematical and Computer modelling*, vol. 58, no. 3, pp. 480–488, 2013.

[7] C. Ren and C. P. Chen, "Decentralized control for second-order uncertain nonlinear multi-agent systems consensus problem based on fuzzy adaptive high-gain observer," in *Systems, Man, and Cybernetics (SMC), 2013 IEEE International Conference on*. IEEE, 2013, pp. 4935–4940.

[8] G. M. P. Wanderley, M. P. Ramos, C. Tacla, G. Y. Sato, E. J. d. Silva, and E. C. Paraiso, "Modus-sd: User modeling in collaborative software development," in *Computer Supported Cooperative Work in Design (CSCWD), 2012 IEEE 16th International Conference on*. IEEE, 2012, pp. 372–378.

[9] N. Negroponte, *Being digital*. Vintage, 1996.

[10] E. C. Paraiso and J.-P. A. Barthès, "An intelligent speech interface for personal assistants in r&d projects," *Expert Systems with Applications*, vol. 31, no. 4, pp. 673–683, 2006.

[11] G. M. P. Wanderley, M.-H. Abel, J.-P. Barthès, and E. C. Paraiso, "An advanced collaborative environment for software development," in *Systems, Man, and Cybernetics (SMC), 2016 IEEE International Conference on*. Accepted, In press, 2016.

[12] T. J. McCabe, "A complexity measure," *Software Engineering, IEEE Transactions on*, no. 4, pp. 308–320, 1976.

[13] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *Software Engineering, IEEE Transactions on*, vol. 20, no. 6, pp. 476–493, 1994.

[14] B. H. Sellers, "Ojbect-oriented metrics. measures of complexity," 1996.

[15] R. C. Martin, *Agile software development: principles, patterns, and practices*. Prentice Hall PTR, 2003.

[16] R. Harrison, S. Counsell, and R. Nithi, "An overview of object-oriented design metrics," in *Software Technology and Engineering Practice, 1997. Proceedings., Eighth IEEE International Workshop on [incorporating Computer Aided Software Engineering]*. IEEE, 1997, pp. 230–235.

[17] J.-P. A. Barthès, "Omasa flexible multi-agent environment for cscwd," *Future Generation Computer Systems*, vol. 27, no. 1, pp. 78–87, 2011.

[18] M.-H. Abel, "Knowledge map-based web platform to facilitate organizational learning return of experiences," *Computers in Human Behavior*, 2014.

[19] Metrics, "Eclipse Metrics Plugin," http://metrics2.sourceforge.net, 2016, online; accessed 15 September 2016.

[20] V. Botti, A. Garrido, A. Giret, and P. Noriega, "The role of mas as a decision support tool in a water-rights market," in *International Conference on Autonomous Agents and Multiagent Systems*. Springer, 2011, pp. 35–49.

[21] M. Bokhari and S. Ahmad, "Design for interactive e-learning based upon multi-agent system: I-mbls," in *Confluence 2013: The Next Generation Information Technology Summit (4th International Conference)*. IET, 2013, pp. 456–460.

[22] H. Ellouzi, H. Ltifi, and M. B. Ayed, "New multi-agent architecture of visual intelligent decision support systems application in the medical field," in *Computer Systems and Applications (AICCSA), 2015 IEEE/ACS 12th International Conference of*. IEEE, 2015, pp. 1–8.

[23] D. E. Robbins, V. P. Gurupur, and J. Tanik, "Information architecture of a clinical decision support system," in *Southeastcon, 2011 Proceedings of IEEE*. IEEE, 2011, pp. 374–378.

[24] D. Polajnar, M. Zubayer, and J. Polajnar, "A multiagent architecture for semantic access to legacy relational databases," in *Systems Conference (SysCon), 2012 IEEE International*. IEEE, 2012, pp. 1–8.