Position Papers of the Federated Conference on Computer Science and Information Systems pp. 247–254
DOI: 10.15439/2016F379
ACSIS, Vol. 9. ISSN 2300-5963

APIS – Agent Platform for Integration of Services

Michał Wójcik
Faculty of Electronics, Telecommunications and Informatics
Gdańsk University of Technology
Gabriela Narutowicza 11/12, 80-233 Gdańsk, Poland
Email: michal.wojcik@eti.pg.gda.pl

Paweł Napieracz and Wojciech Jędruch
Faculty of Electronics, Telecommunications and Informatics
Gdańsk University of Technology
Gabriela Narutowicza 11/12, 80-233 Gdańsk, Poland
Email: wjed@eti.pg.gda.pl

***Abstract*—This paper presents an approach to create a platform for development and evaluation of task execution algorithms relying on services composition. Proposed solution is based on an agent paradigm where autonomous agents can cooperate and negotiate in order to execute specified tasks which are defined by input/output descriptions. Tasks are realized by the means of services exposed by different agents. In case when there is no a single service fulfilling the submitted task requirements, there is a need for an automated composition of services into one complex workflow. The platform provides ready to use communication blocks which can be easily used for algorithms development without consideration for complex conversation protocols handling. All the algorithms developed on the platform are service implementation independent and oriented on inter-agent communication.**

I. Service Composition Problem

THE SERVICE composition is not a new problem and has been already considered in the literature. There is a number of different approaches to this problem:

- Centralized service providers systems which do not provide any kind of composition [1], but only give the user access to different resources in a uniform manner.
- Centralized systems providing workflow static composition and dynamic services selection features [2], [3]. Those require the user to define all the tasks in the workflow and the system itself performs only dynamic services selection based on the requested QoS parameters. They are often focused on the specific services architecture instead of generic algorithms.
- Decentralized agent service providers systems [4]. Those consider mainly broker agents providing features for discovering and negotiating services execution parameters. They do not provide any means of workflows composition but can be used as underlying systems.
- Decentralized agent systems with static workflow composition [5], [6]. In contrast to centralized systems, they can use autonomous agents for dynamic services selection. Agents acting as services brokers can negotiate compatibility and QoS parameters of the services.
- Systems, both centralized and decentralized, with dynamic workflow composition [7], [8]. Those require from the users only definition of the required output and optional input. Decentralized agent systems with dynamic composition are often used in the simulation of business processes used in the virtual organizations.

Most of the services composition systems are focused on the particular services architectures and description standards (UDDI, WSDL, OWL-S). The implemented cooperation algorithms are tested together with services efficiency which does not give the generic knowledge about the algorithms itself. Moreover, agent solutions not always consider agent communication standards which makes them even more limited to a particular solution.

A generic testbed environment, APIS (Agent Platform for Integration of Services), was created. Because the platform is not based on any particular service architecture it allows for testing cooperation algorithms with a focus on their performance rather than on services execution performance. The platform provides means for discovering, negotiating and executing abstract services using inter-agent communication based on the FIPA communication protocols [9].

There is a number of approaches for describing services for the composition process needs. Those can be full ontological descriptions concerning input/output syntactical definitions as well as semantical process definitions and some additional preconditions [10]. This allows for full ontological reasoning about services compatibility as well as desired output. Another known approach is syntactical input/output description combined with semantical service description based on thesaurus allowing for services matching based on words semantical distance [11]. Possibly simpler approach is semantical and syntactical input/output description with only I/O compatibility reasoning [12].

This paper proposes different solution, based on input/output and QoS ontological descriptions allowing for reasoning output $\rightarrow$ input compatibility between services. It assumes that I/O and QoS descriptions are enough for describing what and how should be done. Because this work does not consider different services architectures, at this point services adaptation has not been taken into consideration.

II. Agents as Service Providers

According to the most basic definition, an agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its delegated objectives. It might as well be an environment of some kind of services. Those can be both, Web Services distributed on remote machines connected to

the Internet as well as business services representing company activities mapped into computer system for the sake of simulations and automation. Agents can be treated both, as autonomous services providers and executors existing in such an environment. Moreover multi agent systems which assume communication and interaction between agents residing in the system, are suitable for this case.

When one agent is going to invoke a service of another one, there is a need for some kind of agreement between them to be established. Such an agreement should be made on the basis of some negotiations and be profitable for both sides. This actions can be described by Service Level Agreement (SLA) which is contractual obligations between a service consumer and a service provider, which can represent guarantees of quality of service (QoS), non-functional requirements of a service consumer and promises of a service provider [13].

III. Model of Self-Organization

This section presents a proposal of solution for the tasks composition problem. It distinguishes between different roles which can be taken by composing agents as well as between different communicative acts used in the composition process.

A. Agent Architecture

Figure 1 presents the APIS agent abstract architecture overview. It is a variation of the layered architecture where each of the layer can have a number of sub layers. All the layers (even the sub layers) can perceive input by a means of the see function which basically receives messages from other agents in the environment as well as produce output made of messages directed to those agents. Layers can be spawned dynamically by other layers and be attached as sub-layers or top level ones. All the layers are connected to agent inner state (for the sake of simplification, the architecture figure shows only one such a connection) which basically is a set of a services (both owned and those provided by other agents) known to the agent. Moreover, each of the layers contains its own state which is shared only with parent layer and sub layers. This state allows for performing long running actions based on previous interactions with other agents.

In this model, the only agent interactions with its environment are done through messages exchanged with other residing agents. The environment state can be described as one ore more messages (possibly from different agents) perceived within some context (e.g.: asking about particular service):

$$e = \{\mu_1, \mu_2, \ldots, \mu_n\} \tag{1}$$

where:

- $\mu \in \mathcal{M}$ which is a set of all possible messages.

This leads to defining agents actions also as set of messages (possibly addressed to different receivers):

$$\alpha = \{\mu_1, \mu_2, \ldots, \mu_n\} \tag{2}$$

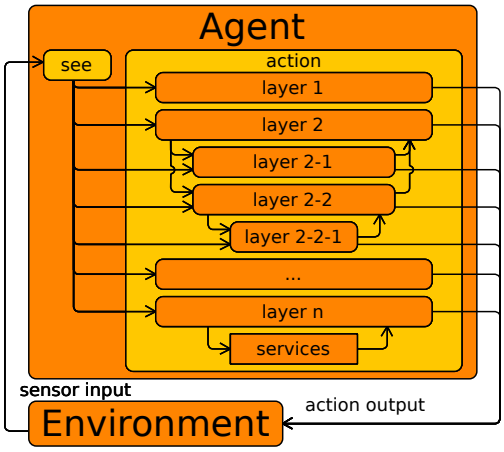

Fig. 1. Agent and its environment in the APIS platform

Finally, the agent decision function can be defined as a mapping from sequences[1] of messages sets[2] to messages sets:

$$a : \wp(\mathcal{M})^* \rightarrow \wp(\mathcal{M}) \tag{3}$$

The agent run function can be defined as subsequent environment state (message set) to action (messages set) transitions:

$$r : \mathcal{M}_0^i \xrightarrow{\mathcal{M}_0^o} \ldots \xrightarrow{\mathcal{M}_{u-1}^r} \mathcal{M}_u^o \tag{4}$$

where:

- $\mathcal{M}^i \in \wp(\mathcal{M})$ is a set of input messages,
- $\mathcal{M}^o \in \wp(\mathcal{M})$ is a set of output messages.

The run function can be formally presented as a mapping of environment states sequences and action sequences to environment states:

$$r : \mathcal{E}^* \times \mathcal{A}^* \rightarrow \mathcal{E} \tag{5}$$

Similarly to agent definition, the standard agent see function mapping environment state sequences to percepts can be defined as a mapping from a set of messages to percepts:

$$see : \wp(\mathcal{M})^* \rightarrow P \tag{6}$$

and the $action$ function mapping sequence of percepts to actions as a mapping from sequences of percepts to a set of messages:

$$action : P^* \rightarrow \wp(\mathcal{M}) \tag{7}$$

In layered architectures, decision function is realized through a set of behaviours, each associated with one layer. Because single layer can take part in ongoing inter agent negotiations, it can produce a number of different actions (messages sent to different agents) as well as be activated by a number of different environment states (messages from different agents) transformed into percepts. Because this is not a traditional layered approach where behaviours are described

[1]Sequences over set S are written as S^*
[2]Power set over set S is written as $\wp(S)$

as a pair of condition set and a resulting action there is a need for additional layer action function which defines how specified inputs are transformed into outputs:

$$beh = (P^c, \mathcal{A}^r, beh\_action) \tag{8}$$

where:

- P^c is set of percepts called the condition,
- $\mathcal{A}^r$ is set of possible actions called the result,
- beh_action is single layer action function.

A single behaviour action function can be defined as a mapping of percepts sequences and services sets to actions (sets of messages):

$$beh\_action : P^* \times \wp(Se) \to \wp(\mathcal{M}) \tag{9}$$

where:

- $Se = \{se_1, se_2, \ldots, se_{|S|}\}$ is a set of all services.

In order to compare different approaches to complex service workflows composition, a number of different utility functions can be introduced. A successful composition utility function returns values 1 and 0 determining if a composition process for a particular task was successful or not:

$$u_s : \mathcal{R} \to \{0, 1\} \tag{10}$$

where:

- $\mathcal{R} = \{r_1, r_2, \ldots, r_{|\mathcal{R}|}\}$ is a set of all possible runs in the environment.

A message utility function gives a natural number telling how many messages were used for a particular task execution:

$$u_m : \mathcal{R} \to \mathbb{N} \tag{11}$$

and a conversations utility function says how many different conversations have been started:

$$u_c : \mathcal{R} \to \mathbb{N} \tag{12}$$

A QoS utility function gives information about values of different QoS parameters describing the composed business process execution:

$$u_q : \mathcal{R} \to \wp(\mathbb{N}) \tag{13}$$

A time utility function tels how much real time (this value can vary on different hardware environment configurations) was used for the composition process:

$$u_t : \mathcal{R} \to \mathbb{R} \tag{14}$$

B. Agent Environment

The APIS platform assumes an agent system where a number of agents are spawned in order to cooperate. The multi-agent system can be formally described as:

$$sys = \langle A, env \rangle \tag{15}$$

where:

- A is a set of all agents in the system,
- $env = \langle \mathcal{E}, e_0, \tau \rangle$ is agent environment with initial state and state transfer function defined.

The classification of the agent environment in the APIS platform can be considered in two different scenarios concerning platform life time:

- short-run – all agents spawned at the same time only for single task execution request,
- long-run – agents can be spawned dynamically during platform lifetime, many independent task execution requests.

The environment characteristics which are common for both situations and does not change depending on platform life time are (based on [14, p. 30]):

- non-deterministic – because agents' actions consequences depend on inner states of all the agents taking part in a interaction, the single result can not be fully predicted,
- dynamic – dynamic environments change without agent interaction, the APIS platform environment can change only as a result of agents' action, but not all of the agents always take part in those interactions so the environment can change without their knowledge.

In the short-run, agents are spawned only for a single task execution and removed form platform afterwards. This method can be used for testing different algorithms and comparing them with different agents and services configurations. Moreover it can be used for cases when particular task should be carried on without any dependencies to other possible tasks. The environment characteristics in this situation are (based on [14, p. 30]):

- episodic – there is no connection between scenarios as agents are spawned only for a single task execution,
- discrete – there is finite number of environment and agents states (especially services composition possibilities) resulting from the initial platform configuration.

In the long-run, agents can be spawned and removed dynamically during the whole platform life-cycle. This allows agents to learn new composed services resulting from different tasks executions. This approach is especially useful in virtual organizations simulations. The environment characteristics in this situation are (based on [14, p. 30]):

- non-episodic – agents' decisions concerning the composition process are based on their knowledge about services, in a long-time running environment agents learn about new services and conditions negotiated at some point can influence future compositions,
- discrete – because number of agents residing on the platform, and services they know can change, there is an infinite number of services composition possibilities.

IV. INFRASTRUCTURE

The developed APIS platform [15], [16] is based on the JADE (Java Agent DEvelopment Framework) which is an agent development framework allowing for creating distributed multi agent systems [17]. It is one for the mostly used and recognizable agent platforms [5], [4], [3], [8], [6].

JADE supports behavior-oriented agent model, that means all agents actions are in a form of behaviors launched during

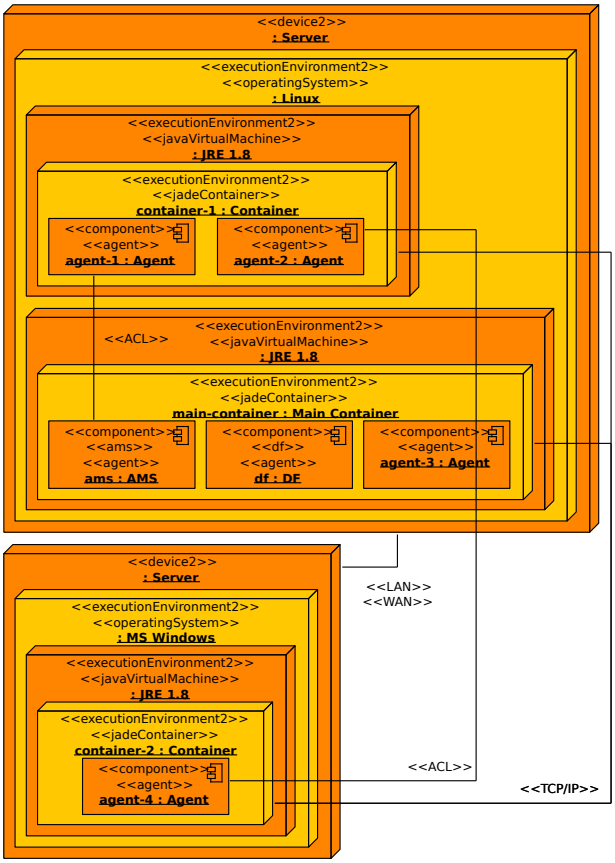

Fig. 2. Example JADE infrastructure

agent life cycle. One agent can make use of a number of different behaviors purposed for realization of different goals. New behaviors can be added while launching agent as well as during its life-cycle from other behaviors. This allows for dynamically adding behaviors related to the decisions made by an agent and clearly complies with the agent model defined in this paper.

An agent environment can be build with several nodes (in JADE called containers) which can run on one or more physical machines connected with network creating distributed environment. All the agents reside in those containers. JADE configuration requires at least one main container responsible for the whole platform, other dependent containers connect to the main one. JADE allows for configuration with a number of backup main containers synchronizing during life-cycle. Figure 2 shows an example JADE infrastructure. Only few connection between agents components were shown for the sake of simplification.

All agents belonging to the same platform can communicate with each other using ACL (Agent Communication Language), a standard language for agents communication defined by FIPA (The Foundation of Intelligent Physical Agents) [9]. Messages content is defined by two things: Semantic Language (SL) defining grammar of the message and domain ontology defining vocabulary. The language is defined by FIPA and is provided in JADE as a SL Codec whereas the ontology must be provided by the developer.

V. Agents Types

When concerning roles in the complex tasks execution process, different approaches can be taken. The APIS platform allows for deploying equal peers without any relations between them as well as agents organized into some hierarchies.

There are different roles, that can be taken by agents in the tasks execution process:

- client – an agent that searches for agents capable of executing particular task,
- principal – an agent that has some subordinates from whom it can request some actions,
- contractor – an agent able to expose some services, their QoS parameters as well as payment conditions,
- coordinator – additional role for contractor, introduced for better readability of centralized algorithms,
- subordinate – an agent that has a principal which can request some actions,
- collaborator – an agents collaborating with other agent on equal rights in order to perform some actions.

Then, different relationships between agents can be listed:

- client - contractor (coordinator) – client searches for contractors capable of executing particular tasks, during negotiations contractor provides QoS parameters which are evaluated by client,
- principal - subordinate – those are agents inside the same agency, where principal belongs to the sub-agency higher in the hierarchy, there are no negotiations and subordinate is not able to refuse performing requested tasks unless it is not capable of performing it,
- collaborator - collaborator – those can be agents in the same agency and both belonging to sub-agencies on the same level in the hierarchy, they can be requested by principal to perform some tasks together and they must jointly work out a solution.

One agent can be in more than one role at the same time. For example the same agent can be contractor for external client, collaborator for agents in the same sub-agency and principal for agents in sub-agencies lower in the hierarchy at the same time.

VI. Communication Protocols

According to the FIPA standard, there is a number of protocols describing in details communication between agents [9]. In order to ensure that the APIS platform is complying with the FIPA standard, all the composition algorithms should be based on the FIPA protocols. Initially, for the needs of this work the five protocols were chosen: cancel, request, query, contract net and iterated contract net protocols.

The cancel meta protocol allows the initiator to cancel on going interaction with another participant under any protocol [9]. The initiator trying to cancel an interaction, needs to send `cancel` message containing the message that it

wants to cancel. The participant can reply with `inform` message after successful termination or `failure` message when termination did not succeed.

The query protocol allows the initiator to ask the participant if a given proposition is true by sending `query-if` message or ask for information concerning a given object by sending `query-ref` message [9]. The participant can agree or not agree to respond by replying with respectively `agree` or `refuse` message. If the participant agreed to response, it sends `inform` message containing true/false reply or information concerning provided object. It can also send `failure` message if an attempt to acquire the answer finished with an error.

The request protocol allows the initiator to request an execution of a given action by the participant by sending a `request` message [9]. The participant can refuse performing the given action by replying with a `refuse` message or agree by replying with an `agree` message. The participant replies with an `inform` message which contains an action outcome. The participant can reply with a `failure` message if performing the action finished with an error.

The contract net protocol allows the initiator to gather a number of proposals of performing some action from one ore more participants [9]. Firstly, the initiator sends a `cfp` message containing action description to potential performers. The participants can reply with a `propose` message containing some conditions of executing the given action or with a `refuse` message when they are not interested. After gathering all the replies or exceeding a deadline (specified in the first `cfp` message) the initiator browses the proposals and selects one or more the best ones. Authors of the selected proposals receive an `accept-proposal` message and authors of the rejected ones receive a `reject-proposal` message. All the proposals received after the given deadline are automatically rejected and their authors receive a `reject-proposal` message with a corresponding information. Similarly to the request protocol, after finishing the action, the participants send an `inform` message which can contain an action result or `failure` message in case of a failure.

The iterated contract net enriches the contract net protocol with a possibility of stating more exact conditions in a negotiation process [9]. After selecting propositions, the initiator can decide if that was or not the final iteration. If it was the final iteration, protocol proceeds as in standard contract net protocol. If it was not the final iteration, the initiator sends more exact `cfp` message. This process can be repeated until the initiator decided, that further negotiations are not required.

VII. Algorithms Building Blocks

The idea of the APIS platform is to develop complex tasks execution algorithms using pre-made block providing all inter-agent communication actions so the developer can focus on the algorithms structure. Normally developer would be forced to implement all the communication stack including packing, unpacking, sending, receiving and filtering messages within a number of ongoing conversations.

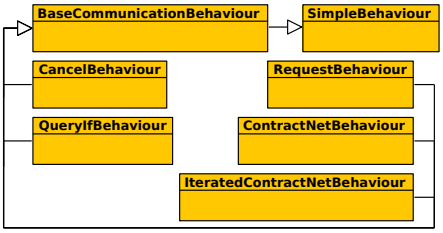

Fig. 3. Active behaviours

Figure 3 presents hierarchy of behaviours which can be used by the developer while creating an active side of the execution process. Those behaviours are:

- `SimpleBehaviour` – basic JADE behaviour class for all agent behaviours,
- `BaseCommunicationBehavior` – basic communication stack operations common for all the used protocols,
- `CancelBehaviour` – implementation of *meta cancel* protocol, allows to cancel any ongoing conversation,
- `QueryIfBehaviour` – implementation of *query if* protocol, allows to check if given fact is true according to other agents, requires only providing the fact and a receiver,
- `RequestBehaviour` – implementation of the *request* protocol, allows to request performing some action by another agent, requires providing the action definition and a receiver,
- `ContractNetBehaviour` – implementation of the *contract net* protocol, allows to call for proposals of performing some action, gather those proposal, select the best one and gather the result, requires providing the action definition, a list of receivers and a proposals comparator,
- `IteratedContractNetBehaviour` – implementation of *iterated contract net* protocol, allows to do the same as the `ContractNetBehaviour` but with negotiation iterations.

Figure 4 presents hierarchy of behaviours which are automatically used by the passive side of the execution process. Those can not be used directly by the developer and are launched automatically by the agent when specified initiating messages is received. Those behaviours are:

- `ResolverBehaviour` – basic resolving behaviour implementing common communication stack,
- `CancelResolverBehaviour` – behaviour launched when a *cancel* message is received,
- `QueryIfResolverBehaviour` – behaviour launched when a *query if* message is received, checks submitted fact and responds with a result,
- `RequestResolverBehaviour` – behaviour launched when a *request* message is received, performs requested action and responds with a result,
- `CallForProposalResolverBehaviour` – behaviour launched when a *call for proposal* message

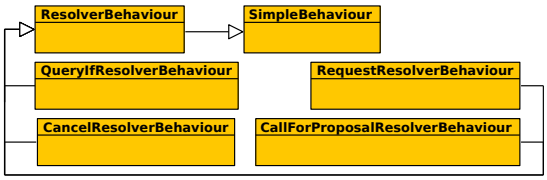

Fig. 4. Passive behaviours

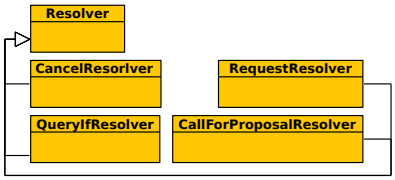

Fig. 5. Passive behaviours resolvers

is received, prepares proposal and if accepted performs request action and responds with a result.

While the passive behaviours are fixed, their outputs can be changed by the mean of resolvers. The resolvers are interfaces which can be implemented by the developer and which are used by the passive behaviours. Figure 5 presents a hierarchy of resolvers used on the platform. Those interfaces are:

- `Resolver` – basic interface for all the resolvers,
- `CancelResolver` – defines actions carried on by the agent after receiving *cancel* message,
- `QueryIfResolver` – defines actions for checking if giver fact is true,
- `RequestResolver` – defines actions of executing specified action and returning its result,
- `CallForProposalResolver` – defines action of preparing proposal and if accepted execution specified action and returning its result.

VIII. Simple Composition Algorithm

In order to show that the platforms fulfills its requirements, the simple composition algorithm was prepared. During its implementation no communication based code was prepared. Figure 6 presents classes which were developed and their relation to those provided by the platform. Those classes are:

- `Coordinator` – agent coordinating the composition process, registers appropriate request resolver,
- `Contractor` – agent providing some services, registers appropriate call for proposal resolver,
- `ReqResolver` – implementation of the request resolver, when receiving a request to execute some task it starts the composition behaviour,
- `CFPResolver` – implementation of the call for proposal resolver, checks if requested task output can be provided by any of the services known by the resolver owner and if yes prepares an appropriate proposal,
- `CompositionBehaviour` – a behaviour responsible for composing a new workflow of services in order to provided desired task output, in order to find subsequent

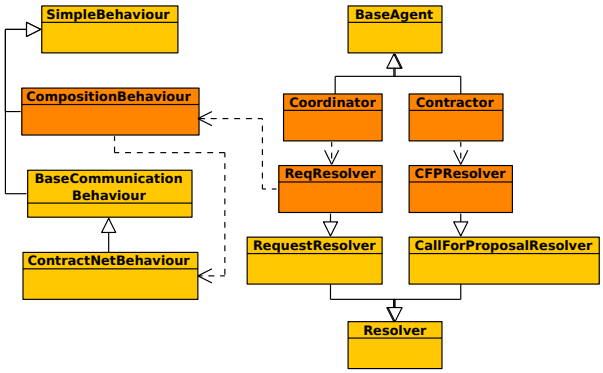

Fig. 6. Simple algorithm

workflow services it starts underlying contract net behaviours.

IX. Running the Algorithm

The designed algorithm was carried on by the following agents exposing specified services:

- agent-pizza-maker:
  - MakePizza : (base, topping, sauce) → (pizza);
- agent-baker:
  - MakeBase : (flour, water) → (base);
- agent-sauce-maker:
  - MakeSauce : (tomato, water) → (sauce);
- agent-topping-maker:
  - MakeTopping : (vegetable) → (topping);
- agent-seller:
  - ProvideFlour : () → (flour),
  - ProvideWater : () → (water),
  - ProvideVegetable : () → (vegetable),
  - ProvideTomato : () → (tomato);
- agent-coordinator,
- testRunner.

All the agents with the *agent-* prefix are contractors without any hierarchical relationships. The agent-coordinator is an agent which receives a request from the client (the testRunner agent). Only services exposed by the agent-seller agent do not require any input so they should be used as the workflow initial services.

The subsequent messages exchanged by the agent are presented in figures from 8 to 11 which were created with APIS version of JADE sniffer agent. The communication snapshot presents which messages belong to which conversation. The explanation of goals of each conversation is presented in table I. The final workflow providing desired output is presented in figure 7 which was created using the APIS service sniffing tool. Values for the utility (10, 11, 12, 13, 14) functions are: $u_s = 1$, $u_m = 129$, $u_c = 55$, $u_q = 9$, $u_t = 150.5$ms.

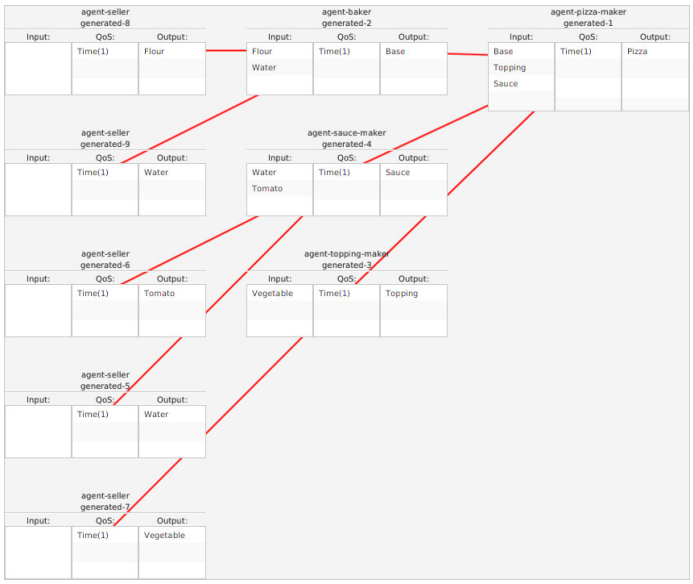

Fig. 7. Service *generated-0* chain for simple centralized algorithm

TABLE I
CONVERSATIONS SUMMARY FOR SIMPLE CENTRALIZED ALGORITHM

conversations	protocol	output	service
0	request	pizza	generated-0
1, 7, 13, 19, 25, 31, 37, 43, 49	request	agents	-
2, 3, 4, 5, 6	cfp	pizza	generated-1
8, 9, 10, 11, 12	cfp	base	generated-2
14, 15, 16, 17, 18	cfp	topping	generated-3
20, 21, 22, 23, 24	cfp	sauce	generated-4
26, 27, 28, 29, 30	cfp	water	generated-5
32, 33, 34, 35, 36	cfp	tomato	generated-6
38, 39, 40, 41, 42	cfp	vegetable	generated-7
44, 45, 46, 47, 48	cfp	flour	generated-8
50, 51, 52, 53, 54	cfp	water	generated-9

X. CONCLUSION

The idea of this work was to provide an agent platform allowing for developing and evaluating complex tasks execution and services composition algorithms. In order to focus on the algorithms, the platform is not based on any services implementation but only on inter agent FIPA communication standard. It has been shown how APIS algorithms building blocks comply with the FIPA communication protocols and that they can be successfully used in developing execution and composition algorithm. The set of building blocks can be easily expanded with new protocols by implementing low-level communication stack. Moreover it has been shown that platform accompanying sniffing tools allow for good algorithms evaluation. Proposed utility functions can be used for comparing different algorithms.

Despite the fact that the APIS platform is not based on any services implementation it can be in future easily enriched with one by changing appropriate algorithms building blocks. As future work, the most important possibility is developing and evaluating more complex execution and composition algorithms, especially distributed ones based on work division.

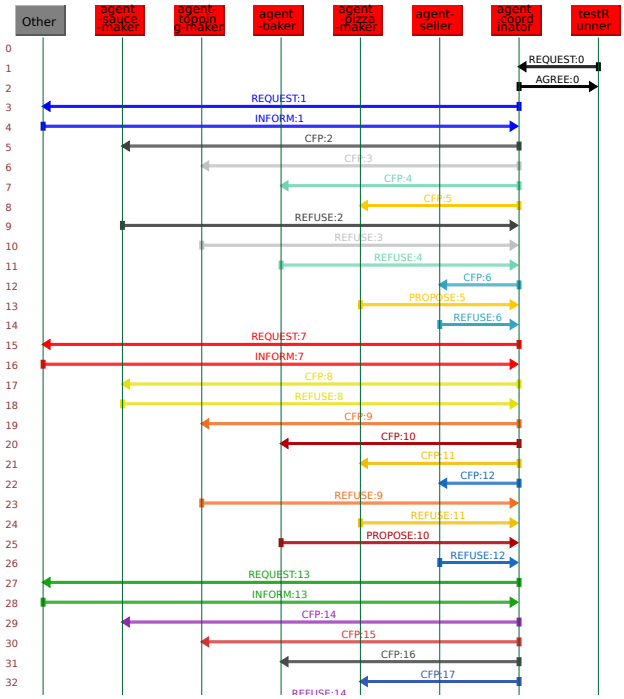

Fig. 8. Communication snapshot

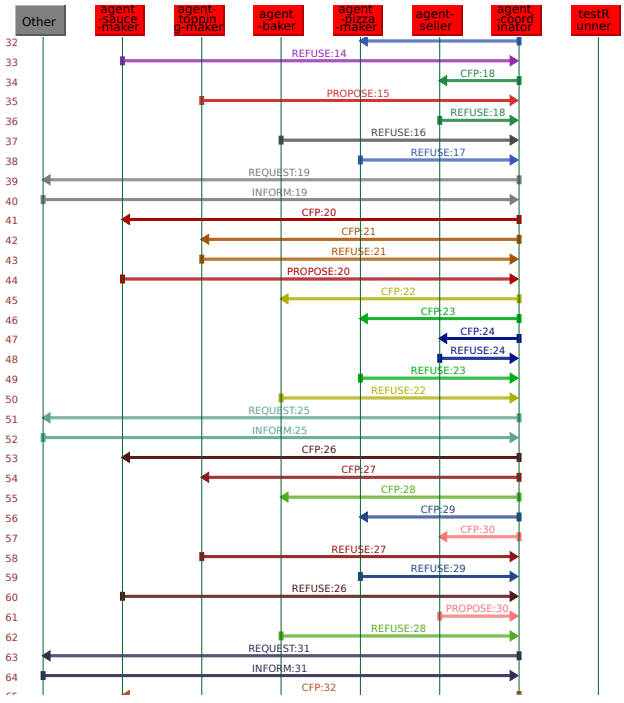

Fig. 9. Communication snapshot (continued)

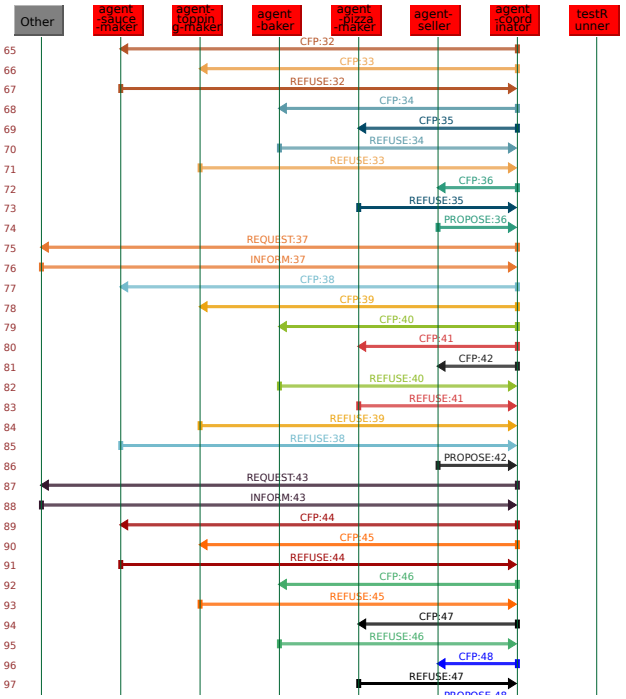

Fig. 10. Communication snapshot (continued)

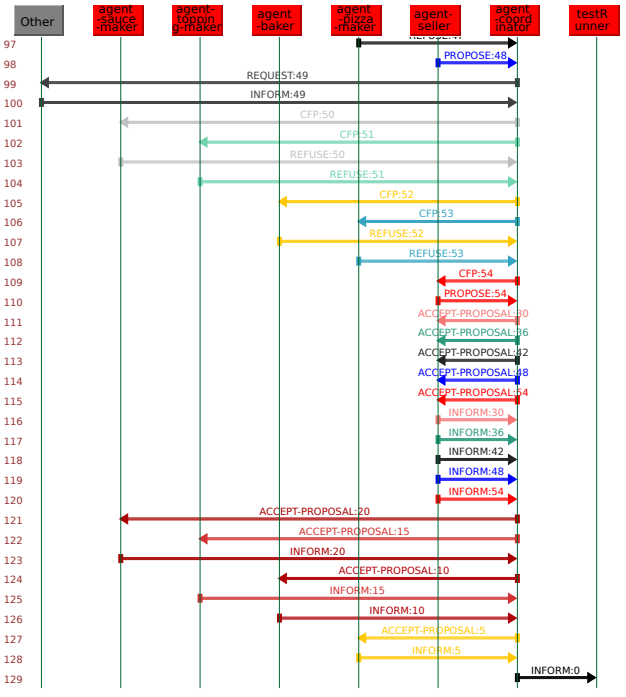

Fig. 11. Communication snapshot (continued)

REFERENCES

[1] P. Czarnul, "A JEE-Based Modelling and Execution Environment for Workflow Applications with Just-in-Time Service Selection," in *Proceedings of the 2009 Workshops at the Grid and Pervasive Computing Conference (GPC)*, ser. GPC '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 50–57. [Online]. Available: http://dx.doi.org/10.1109/GPC.2009.24

[2] ——, "Comparison of selected algorithms for scheduling workflow applications with dynamically changing service availability," *Journal of Zhejiang University SCIENCE C*, vol. 15, no. 6, pp. 401–422, 2014. [Online]. Available: http://dx.doi.org/10.1631/jzus.C1300270

[3] F.-S. Hsieh and J.-B. Lin, "Context-aware workflow management for virtual enterprises based on coordination of agents," *Journal of Intelligent Manufacturing*, vol. 25, no. 3, pp. 393–412, 2014. [Online]. Available: http://dx.doi.org/10.1007/s10845-012-0688-8

[4] L. Ehrler, M. Fleurke, M. Purvis, B. Tony, and R. Savarimuthu, "AgentBased Workflow Management Systems (WfMSs): JBees - A Distributed and Adaptive WfMS with Monitoring and Controlling Capabilities," in *Journal of Information Systems and e-Business Management, Volume 4, Issue 1*. Springer-Verlag, 2005, pp. 5–23. [Online]. Available: http://dx.doi.org/10.1007/s10257-005-0010-9

[5] P. Czarnul, M. Matuszek, M. Wójcik, and K. Zalewski, "Beesybees: A mobile agent-based middleware for a reliable and secure execution of service-based workflow applications in beesycluster," in *Multiagent and Grid Systems*. IOS Press, 2011, vol. 7, pp. 219 – 241. [Online]. Available: http://dx.doi.org/10.3233/MGS-2011-0178

[6] P. Czarnul and M. Wójcik, "Dynamic compatibility matching of services for distributed workflow execution," in *Parallel Processing and Applied Mathematics*, ser. Lecture Notes in Computer Science, R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Wasniewski, Eds. Springer Berlin / Heidelberg, 2012, vol. 7204, pp. 151–160. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-31500-8_16

[7] F. E. Tosta, V. Braganholo, L. Murta, and M. Mattoso, "Improving workflow design by mining reusable tasks," *Journal of the Brazilian Computer Society*, vol. 21, no. 1, pp. 1–16, 2015. [Online]. Available: http://dx.doi.org/10.1186/s13173-015-0035-y

[8] F.-S. Hsieh and J.-B. Lin, "A self-adaptation scheme for workflow management in multi-agent systems," *Journal of Intelligent Manufacturing*, vol. 27, no. 1, pp. 131–148, 2016. [Online]. Available: http://dx.doi.org/10.1007/s10845-013-0818-y

[9] The Fundation of Intelligent Physical Agents, "FIPA specifications," Tech. Rep., 2002. [Online]. Available: http://www.fipa.org/repository/standardspecs.html

[10] K. Sycara, M. Paolucci, A. Ankolekar, and N. Srinivasan, "Automated discovery, interaction and composition of semantic web services," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 1, no. 1, pp. 27 – 46, 2003. [Online]. Available: http://dx.doi.org/10.1016/j.websem.2003.07.002

[11] K. Arisha, F. Ozcan, R. Ross, S. Kraus, and V. S. Subrahmanian, "Impact: the interactive maryland platform for agents collaborating together," in *Multi Agent Systems, 1998. Proceedings. International Conference on*, Jul 1998, pp. 385–386. [Online]. Available: http://dx.doi.org/10.1109/ICMAS.1998.699225

[12] G. Wickler and A. Tate, "Capability representations for brokering: A survey," in *Available from: www.aiai.ed.ac.uk/ âĹij oplan/cdl/cdl-ker.ps*, 1998. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.367.9865

[13] Q. He, J. Yan, R. Kowalczyk, H. Jin, and Y. Yang, "Lifetime service level agreement management with autonomous agents for services provision," *Inf. Sci.*, vol. 179, no. 15, pp. 2591–2605, Jul. 2009. [Online]. Available: http://dx.doi.org/10.1016/j.ins.2009.01.037

[14] S. J. Russell and P. Norvig, *Artificial Intelligence a modern approach*, 2nd ed. Upper Saddle River, N.J.: Prentice Hall, 2003.

[15] P. Napieracz, "Porównanie agentowych algorytmów kooperacji w wykonywaniu złożonych zadań," Master's thesis, Politechnika Gdańska, Wydział Elektroniki, Telekomunikacji i Informatyki, 2014.

[16] M. Wójcik, "Raport techniczny nr 2/2015: Projekt platformy apis (agent platform for integration of services)," Gdańsk University of Technology, Faculty of Electronics, Telecommunications and Informatics, Tech. Rep., 2015.

[17] F. L. Bellifemine, G. Caire, and D. Greenwood, *Developing Multi-Agent Systems with JADE*. Wiley, 2007.