Scenario Description Language in Multi-agent Simulation System

Armin Stranjak[1], Igor Čavrak[2], and Mario Žagar[2]

[1] Software Centre of Excellence, Rolls-Royce plc, PO Box 31, Derby DE24 8BJ, UK
[2] University of Zagreb, Faculty of Electrical Engineering and Computing, Unska 3, HR-10000 Zagreb, Croatia
armin.stranjak@rolls-royce.com, {igor.cavrak,mario.zagar}@fer.hr

Abstract. Multi-agent system based simulation is used in a number of different areas ranging from modelling of social and economic to technical complex and dynamic systems. In this paper we analyse feasibility of using scenario description language and platform in design and implementation of MAS-based models. A case study is presented where the SDLMAS framework is used to aid in design and implementation of a complex Aero Repair and Overhaul simulator.

Keywords: Multi-Agent System, Interaction, Scenario, Language, Simulation, Aero Repair and Overhaul.

1 Introduction

Multi-agent system (MAS) based simulations are being used in a variety of domains ranging from manufacturing, supply chains, artificial financial markets, electric power markets, modelling social behaviour, sociology and political science. MAS simulations are especially suitable for large scale complex and dynamic systems whose parameters change frequently during run-time which makes analytical methods formidable to use. MAS-based models and simulations efficiently incorporate various resource constraints, dynamic environment, frequently opposing goals of disparate process participants, and ensure satisfying confidence into simulation outcomes. The increased interest into exploitation of agent-based simulations resulted in development of numerous tools and frameworks such as MASON [8], Swarm [9], Repast [10] and NetLogo [13].

MAS-based models are characterised by direct mapping between the entities from the problem (modelled) domain and the model elements representing them. Entities within the target system (simulation) and interactions between them are, in majority of cases, explicitly and individually represented in the model. Such an approach proves beneficial such that: (a) it facilitates the process of transferring concepts and ideas from the problem domain to the modelling (solution) domain, (b) it eases the model validation and verification by (partially) allowing easier mapping of modelled behaviour to particular model components, and (c) it facilitates analysis and interpretation of simulation results to modellers and domain experts.

J. O'Shea et al. (Eds.): KES-AMSTA 2011, LNAI 6682, pp. 169–179, 2011.

In this paper we devote a particular interest to a class of MAS-based simulations with properties of (a) having a number of well-defined roles (agent types) characterised by role-specific behaviour and goals, (b) complex behaviours defined by those roles, (c) a moderate to large number of model agents playing one of the defined roles, and (d) complex and explicit interactions.

Such MAS-based simulation systems share a significant portion of design and implementation phase properties with 'regular' multi-agent systems. It is thus a reasonable assumption that such systems can be based on general-purpose multi-agent platforms or frameworks such as JADE [7], Cougaar [3] etc. In order to facilitate such implementations, a number of tools can be used to address various aspects of such approach.

One of the most important, time consuming and error-prone aspects is the design and implementation of complex and explicit interactions among model-constituting roles (and consequently agents playing those roles within a concrete simulation instance). To alleviate the interaction design complexity problem, a scenario-based approach is advocated by a number of researchers such as IOM/T language [4], Q language [6], interaction protocol modelling framework [12], MAP language [17] and others. Mentioned approaches create an explicit representation of interactions between agents with a varying degree of independence from the target agent platform, implementation language and the implementation of concrete role (agent) logic.

The SDLMAS (Scenario Description Language for Multi-Agent Systems) platform [2][14] has been devised in order to support rapid design, development and deployment of multi-agent systems. The declarative language in conjunction with the corresponding development and run-time framework forms the core components of the SDLMAS platform.

In the rest of the paper, Section 2 presents the main properties of the SDLMAS language and platform, characteristics that make SDLMAS suitable for design and implementation of MAS-based simulations. Section 3 presents a case study of SDLMAS application in design and implementation of an Aero Repair and Overhaul simulator. Section 4 concludes the paper.

2 SDLMAS

2.1 Scenario Description Language

The SDLMAS language is a declarative, interaction-centric description language, designed with the purpose of defining permitted sequences of messages (communicative acts) exchanged among interaction participants (agents in a multi-agent system). The language defines an interaction protocol implicitly, as a series of actions where protocol definition is achieved through a sequential approach of agent actions. Conversations among agents are described as a sequence of conversation actions, where actions define a conditional mapping between incoming and outgoing messages, and an agent's internal logic. Conditions for reception and transmission of messages are defined explicitly as a part of a conversation

protocol definition. A procedure is defined as an elementary action of the language, only to be executed as a result of a condition being satisfied following receipt of one or more messages.

A scenario consists of a series of conversation actions describing a conversation protocols amongst agents. The language addresses only the communication aspect of multi-agent systems, and therefore imposes strict division between agent's internal business logic and conversation actions. Developed scenarios are independent of the agent platform and implementation language. This is due to language's declarative nature and strict separation of communication aspects of multi-agent system from implementation of agents' internal business logic.

Agent types or roles, agent references and a set of interaction scenarios are the main elements for conversational behaviour description of multi-agent systems using SDLMAS language. A scenario description has a textual format where the header part contains the definitions of agent types and agent references, while the rest of the body contains a set of scenarios that the system in question needs to perform. An agent role is defined as a standardised pattern of behaviour required by all agents participating in conversations. SDLMAS utilises an implicit approach to role behaviour definitions by defining a number of roles that belong to conversations actions within all defined scenarios.

Conversation actions are defined within the scope of a scenario and are attached to agent roles. Each conversation action defines a procedure and two communication conditions: precondition and postcondition (with exception of first and last scenario action). The procedure represents an internal agent logic function, and is invoked by the SDLMAS runtime framework upon satisfaction of communicative preconditions. Communicative preconditions and postconditions are defined with respect to message performative(s) and message originating agent role(s). A communicative precondition defines circumstances under which a received message or a set of received messages are being passed to an internal procedure implementing agent logic. A condition is defined as a list of expected message performatives and their originating agent roles, and can form expressions using logical operators. Communicative postconditions ensure that all messages generated by agent's internal procedure conform to message transmission conditions.

Agent references correspond to a group of agents attached to a specific role. All references are declared before scenario description definitions of SDLMAS file. They are characterised by their role, cardinality and binding method. There are five agent reference types identified: verbatim, variable, anonymous, role group and variable group references.

2.2 SDLMAS Platform

SDLMAS platform provides tools and a framework for implementation and runtime support of multi-agent systems whose interactions are modelled using SDLMAS language. Several components make a set of core elements of this platform: Management Agent, responsible for system initialisation, configuration and system monitoring; Application agents, developed agents with a support of

SDLMAS language and corresponding code generator; Naming Service Agent, responsible to provide subscribe/publish service for agent discovery; underlying target agent platform components including ORB related libraries, etc.

The SDLMAS platform relies on core functionality provided by the target agent platform such as agent lifecycle management, FIPA compliant messaging, agent container management, naming service as a central storage of agent references, etc. Upon their successful initialization by the Management Agent, all agents are required to register with the Naming Service Agent, and to deregister prior to their deactivation.

Behaviour of an application agent is defined in the SDLMAS scenario description. Programme code generated from the scenario description guarantees conformance to the agent's given role in the system. Code skeletons are generated automatically, while portions of agent code related to the agent's internal logic should be manually developed.

Based on system configuration and scenario descriptions, the Management Agent creates and runs a number of Application agents. After they are initialised and running, Application agents register with the Naming Service and subscribe to other system agents, as determined by the scenario description. Finally, a number of scenarios are initiated by the agents, according to the SDLMAS system scenario descriptions.

2.3 SDLMAS and MAS-Based Simulations

Galan et al. [5] describe a process for designing, implementing and using an agent-based model by identifying four distinct roles and their activities and interactions. This process is used to identify potential points of introducing errors and artefacts. Most of the errors and assumptions are a result of incomplete information passing between process roles and errors in processes of abstraction, approximation or implementation, depending on a role.

SDLMAS framework addresses some of the issues comprising activities performed by a modeller, computer scientist and programmer and interactions between them, thus lowering the risk of miscommunication and implementation errors: (a) it facilitates cross-domain communication (a modeller is seldom from the computing domain) by using a simple declarative description language, thus preventing introduction of interaction-related errors in the process of model approximation; (b) it prevents creation of ambiguous or defect scenario specifications by providing means for formal verification of scenario properties (completeness, liveness etc.); (c) it eliminates errors in the process of model implementation (interaction aspect), drastically reducing implementation time and allowing programmer to focus on business logic aspect of the implementation; and (d) allows easier portability among agent platforms, computer platforms and implementation languages, allowing alternative implementations of the same executable model as a mechanism for detection of artefacts and implementation errors.

However, usage of SDLMAS framework introduces a new type of implicit assumptions related to specifics of runtime semantics; specifically the way multiple

and parallel interactions are handled by specific agent implementations. Such assumptions are handled by metadata specifications determining the execution semantics of scenario actions.

3 Modelling of Aero Engine Scheduling Scenario

With a constant increase of deployed aero engines, Aero Repair and Overhaul (AR&O) industry also needs to scale and improve in order to meet ever-growing need for efficient and effective engine scheduling and servicing. The main challenge lies in meeting the following requirements: (a) timely prediction of engines need for repair in order to satisfy the cost, risk and revenue trade-offs; (b) maximum utilisation of overhaul facilities and minimum of aircraft-on-ground occurrences due to engine unavailability; (c) re-scheduling mechanism to achieve high robustness against unforeseen events that can cause major operation and schedule disruptions; (d) long-term simulated visibility of aero repair and scheduling decisions and major effects of business decisions.

Inherently, AR&O business is greatly versatile that is characterised by large numbers of airline operators, engine fleets, overhaul bases, logistics, etc. that contributes to the non-trivial interaction relationships among them. Such multi-dimensional and complex environment needs to be exploited through modelling and simulations in order to be comprehensively understood and analysed. Simulation of complex systems in the aerospace arena has been proved as a powerful vehicle for rapid prototyping of such systems [1][11], including large-scale research programmes like VIVACE [16]. Simulation of complex systems can be achieved using different approaches such as discrete event modelling, system dynamics, Monte Carlo methods, and finally agent-based simulations. Among these, agents have been shown as the most prominent one with the desirable properties [18].

3.1 Aero Repair and Overhaul System Components

The main functional components of the aero repair and overhaul business are fleet managers, overhaul bases and fleet planner. A Fleet Manager role is, during service, to record engine utilisation and capture engine events such as changes in performance, vibrations, temperature, any sort of irregularities, etc. Statistics such as the number of flights (or cycles) and hours in service, estimation and calculation of reliability of individual critical components and whole-engine reliability is also recorded. A fleet is defined by a set of engines of the same type belonging to a certain airline operator and one particular fleet manager is dedicated to each fleet. An Overhaul Base role is to perform engine repair and maintenance activities. It usually contains several repair lines capable of repairing as many engine types. A Fleet Planner role is to generate overhaul base visit schedules on-demand based on current estimation of remaining engine life. The estimation of a date in the future when an engine needs to be serviced can be determined based on current utilisation rate and allowed number of cycles. It

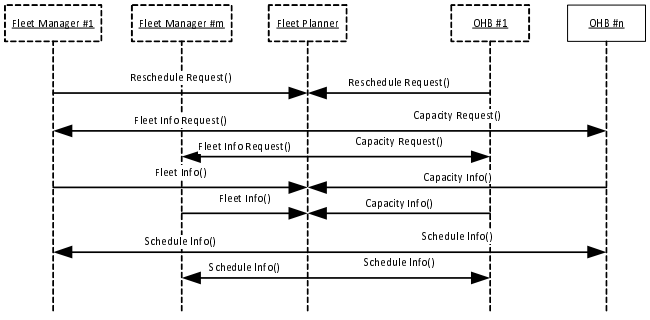

Fig. 1. Rescheduling Scenario

also can be based on a prediction (via continuous monitoring) about when the conditions of engine components reach certain acceptable limits or margins of certain parameters. On top of these, occurrence of an unforeseen event could cause changes of removal date, or indeed induce prompt engine removal and consequent repair. Other factors that the Fleet Planner considers are engine flight pattern and overhaul base capacity, capability and availability. At the same time, the main aim of the Fleet Planner is to maximise revenue by keeping engines in service as long as it is acceptable.

3.2 Scheduling Scenario and Simulator

Scheduling scenario is shown in Figure 1. There are two ways how engine overhaul visit schedule creation can be initiated: (a) on request of fleet managers, or (b) on request of overhaul bases. Either one of these will initiate rescheduling by sending Reschedule Request message.

As soon as Fleet Manager, responsible for certain engine fleet, realises that a particular engine from its fleet needs to be serviced in the period within Schedule Time Horizon (STH) (which is speculative parameter of schedule "visibility" in the future), it will send Reschedule Request message to the Fleet Planner. In other words, if engine's removal date lies within this period, Fleet Manager will request new schedule to be created. The removal date can alter due to: (a) normal engine operation where maximum number of acceptable flights before servicing is estimated to be within STH; (b) changes in an engine's usage pattern or utilisation rate will "bring" engine within STH; (c) fleet-wide updates of engine characteristics will require engines removal in the near future (and within STH); (d) an unforeseen event such as a bird strike that will cause imminent engine removal for servicing.

Overhaul Base would initiate rescheduling operation if there are certain changes in the normal overhaul operation. Typical obstructing events are: (a) delays in current repair operation when engine will not be repaired in time and it potentially can "consume" other engine's repair time; (b) maintenance activities when parts of or entire overhaul base is closed down; (c) unforeseen events such as flood or fire, when normal repair operations are obstructed or cancelled.

```
 1 agent @fleetManager : FleetManager;
 2 agent @ohb          : OHB;
 3 agent fleetPlanner  : FleetPlanner;
 4
 5 scenario Rescheduling {
 6   action FleetManager in rescheduleRequest1() {
 7     msgSnd : (RESCHEDULE_REQUEST -> fleetPlanner);
 8   }
 9
10   action OHB in rescheduleRequest2() {
11     msgSnd : (RESCHEDULE_REQUEST -> fleetPlanner);
12   }
13
14   action FleetPlanner in rcvRescheduleRequest() [timeout=2000]{
15     msgRcv : (RESCHEDULE_REQUEST <- @fleetManager) |
16              (RESCHEDULE_REQUEST <- @ohb);
17     msgSnd : (FLEET_INFO_REQUEST | #BUSY -> <FleetManager>) &
18              (CAPACITY_REQUEST | #BUSY -> <OHB>);
19   }
20
21   action FleetManager in rcvFleetInfoRequest () {
22     msgRcv : (FLEET_INFO_REQUEST | #BUSY <- fleetPlanner);
23     msgSnd : (FLEET_INFO -> fleetPlanner);
24   }
25
26   action OHB in rcvCapacityRequest () {
27     msgRcv : (CAPACITY_REQUEST | #BUSY <- fleetPlanner);
28     msgSnd : (CAPACITY_INFO -> fleetPlanner);
29   }
30
31   action FleetPlanner in generateSchedule () {
32     msgRcv : (FLEET_INFO <- <FleetManager>) &
33              (CAPACITY_INFO <- <OHB>);
34     msgSnd : (SCHEDULE_INFO | #BUSY -> <FleetManager>) &
35              (SCHEDULE_INFO | #BUSY -> <OHB>);
36   }
37 }
```

Fig. 2. Rescheduling SDLMAS scenario definition

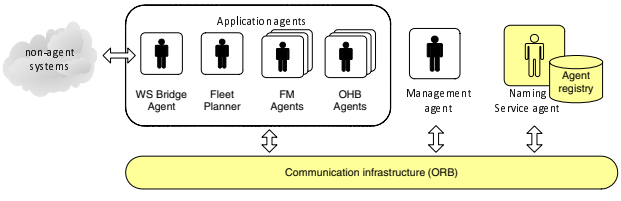

Fig. 3. AR&O Simulator Architecture

After it receives such requests for rescheduling operation, Fleet Planner responds with request for fleet-related and capacity information from fleet managers and overhaul bases (Fleet Info and Capacity requests). Acquired information will enable Fleet Planner to create a new schedule and communicate it to all affected fleet managers and overhaul bases.

The scenario can formally be described in the SDLMAS language, as shown on Figure 2. `FleetPlanner` agent (line 3) is declared using verbatim reference since there is only one fleet planner in the system with already given name (`fleetPlanner`) before runtime. `FleetManager` and `OHB` agent types are used to define roles (line 1-2) of anonymous references to fleet managers and overhaul bases (`@fleetManager` and `@ohb`), whose exact names will be associated with them during runtime. Rescheduling scenario definition (line 5) consists of series of agent actions. A scenario can be initiated by scheduling request from either fleet managers (lines 6-8) or overhaul bases (lines 10-12) to the fleet planner. Upon arrival of scheduling request, fleet planner will request information from

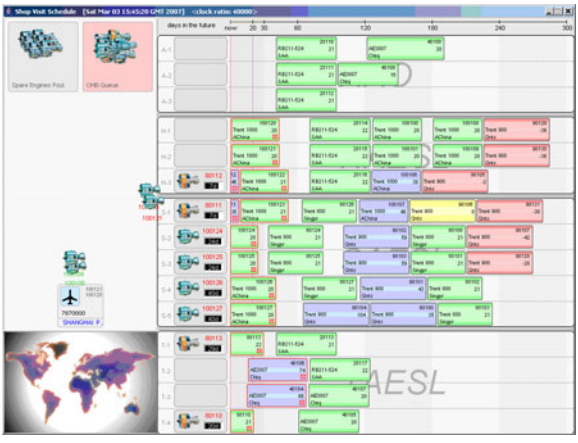

Fig. 4. AR&O Simulator GUI

all fleet managers (`<FleetManager>`) and overhaul bases (`<OHB>`) in order to retrieve fleet-related and overhaul capacity information (lines 14-19). Alternatively, fleet planner can reply with #BUSY terminating message performative, which would indicate that cannot deal with the rescheduling request at that time. After receiving request message, fleet managers and overhaul bases will reply with the fleet (line 21-24) and capacity information respectively (lines 26-29). Finally, fleet planner will create a new schedule and communicate it back to the fleet managers and overhaul bases (lines 31-36).

The aero repair and overhaul simulator architecture, based on SDLMAS platform is shown in Figure 3. Apart from `Management` and `Naming Service` agents, which are part of the SDLMAS platform, `FleetPlanner`, `FleetManager` and `OHB` agents are generated automatically with only internal business logic developed manually. The code is generated using code generators from the SDLMAS scenario description into Java code. Code generation details are outside of scope of this paper and more details can be found in [2]. JADE [7] platform was selected as a base platform (which the SDLMAS platform is built on top of) to support agent creation, communication and life-cycle management. The choice of JADE platform was guided by its FIPA compliance, flexible and open architecture and well-established platform among agent community. `WS Bridge Agent` is developed as an interface to live fleet and engine data through Web Services.

The developed simulator (Figure 4) [15] is capable of simulating several decades of simulated time (one simulated day is roughly equivalent to a few seconds of real time) of engine service utilisation and overhaul visits. Such long simulations are necessary since normal engine lifetime can span over several decades. Analysis of multidimensional parameter space and their influence to overall attributes such as revenue stream, costs, availability, quality of service, etc. is highly complex and convoluted task. Engine utilisation, different environment conditions, OHB availability and capability, application of anticipated

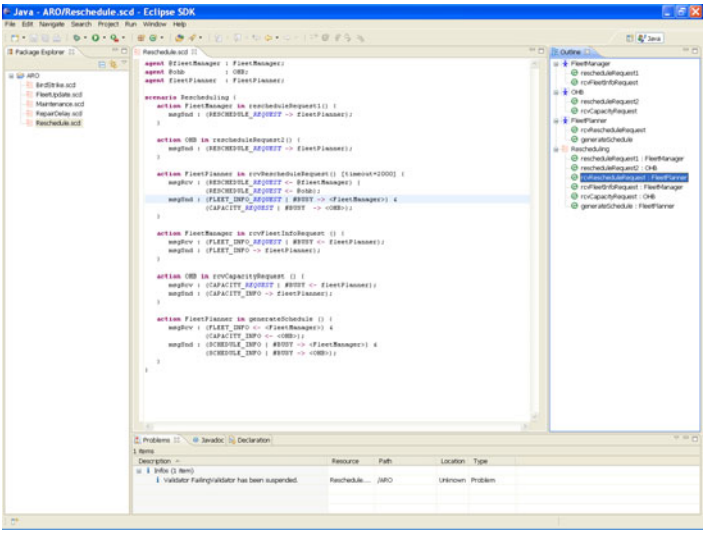

Fig. 5. SDLMAS tool plugin

statistical distribution of unforeseen events, engine reliability and risk estimation models are parameters that have significant influence into the estimation of the future date when an engine needs to be serviced. Only feasible way to analyse this multidimensional space and understand correlations between them is via "what-if" analysis. Such analysis would pre-configure set of parameters and capture interesting system values after every simulation. This way, it is possible, though indirectly, to find correlation matrices between e.g. number of engines in the fleet (including spare engines for quick swaps) and repair turnaround time or OHB utilisation, or number of "aircraft-on-ground" incidents when aircraft cannot fly due to lack of available engines.

The role of SDLMAS language for description of rescheduling scenario and SDLMAS platform are pivotal for the design and development of this simulator. Scenario description helps to design conversational interactions between agents and enables automatic code generation (Figure 5). Majority of code is generated and the developer is required to populate generated code skeletons with agent reasoning logic to make decisions. Furthermore, SDLMAS platform enables entire development and runtime framework for the simulator with minimum configuration and setup tasks. Our estimation is that design and development of the simulator is reduced by factor of 3 if compared with the manual design and development of agent interactions. Also, as a result of usage of code generator, the system is inevitably more robust and stable.

4 Conclusions and Future Work

In this paper we address the field of multi-agent system based simulations, inherent properties found in such systems and focus on a particular subclass of

simulations with implicit and complex interactions among constituent model elements. A SDLMAS framework, consisting of agent scenario description language, tools and a runtime platform is presented and its capabilities in the field of MAS-based simulations analysed. Finally, an agent-based Aero Repair and Overhaul simulator is presented as a proof-of-concept that the SDLMAS framework can successfully address certain issues in design and implementation of complex MAS-based simulations.

The future work on SDLMAS is directed towards enhancing the general capabilities of the language and platform, primarily in enhancement of control over runtime conversation context handling (scenario loops, context concatenation and nesting) and runtime semantics (re-entrant scenario actions, parallel conversation handling).

References

1. Agarwal, R., Gilmer, J.: Back from the future: Using projected futures from recursive simulation to aid decision making in the present. In: Proceedings of the International Symposium on Military, Government and Aerospace Simulation (2004)
2. Cavrak, I., Stranjak, A., Zagar, M.: SDLMAS: A Scenario Modelling Framework for Multi-Agent Systems. Journal of the UCS 15(4), 898–925 (2009)
3. Cougaar: Cognitive Agent Architecture, `http://www.cougaar.org/`
4. Doi, T., Tahara, Y., Honiden, S.: IOM/T: An Interaction Description Language for Multi-Agent Systems. In: 4^{th} International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 778–785. ACM, New York (2005)
5. Galán, J.M., Izquierdo, L.R., Izquierdo, S.S., Santos, J.I., del Olmo, R., Lopez-Paredes, A., Edmonds, B.: Errors and Artefacts in Agent-Based Modelling. Journal of Artificial Societies and Social Simulation 12(1) (2009)
6. Ishida, T.Q.: A Scenario Description Language for Interactive Agents. IEEE Computer 35(11) (2002)
7. Java Agent Development Framework, `http://jade.cselt.it/`
8. Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K., Balan, G.: MASON: A multi-agent simulation environment. Simulation 81, 517–527 (2005)
9. Minar, C.L.N., Burkhart, R., Askenazi, M.: The swarm simulation system: A toolkit for building multi-agent simulations. Tech. report, Sante Fe Institute (1996)
10. North, M.J., Collier, N.T., Vos, J.R.: Experiences creating three implementations of the Repast agent modeling toolkit. ACM Transactions on Modeling and Computer Simulation 16(1), 1–25 (2006)
11. Pritchett, A.R., van Paassen, M.M., Wieland, F.P., Johnson, E.N.: Aerospace vehicle and air traffc simulation. In: Applied System Simulation: Methodologies and Applications, pp. 365–389. Kluwer Academic, Dordrecht (2003)
12. Quenum, J.G., Aknine, S., Briot, J.-P., Honiden, S.: A modeling framework for generic agent interaction protocols. In: Baldoni, M., Endriss, U. (eds.) DALT 2006. LNCS (LNAI), vol. 4327, pp. 207–224. Springer, Heidelberg (2006)
13. Sklar, E.: Software Review: NetLogo, a Multi-agent Simulation Environment. Artificial Life 13, 303–311 (2007)
14. Stranjak, A., Čavrak, I., Žagar, M.: Scenario Description Language for Multi-agent Systems. In: Nguyen, N.T., Borzemski, L., Grzech, A., Ali, M. (eds.) IEA/AIE 2008. LNCS (LNAI), vol. 5027, pp. 855–864. Springer, Heidelberg (2008)

15. Stranjak, A., Dutta, P.S., Ebden, M., Rogers, A., Vytelingum, P.: A Multi-Agent Simulation System for Prediction and Scheduling of Aero Engine Overhaul. In: Proc. 7^{th} International Conference on Autonomous Agents and Multiagent Systems: industrial track (AAMAS 2008), Estoril, Portugal, pp. 81–88 (2008)
16. Value Improvement through a Virtual Aeronautical Collaborative Enterprise (VIVACE), `http://www.vivaceproject.com`
17. Walton, C.D.: Multi-Agent Dialogue Protocols (2003)
18. Yu., T.-T.: A Comparison of Agent-based Modelling with Traditional Discrete-event Modelling. Report ECSTR-LSL07-006, University of Southampton (2007)