An Approach to Designing Belief-Desire-Intention Based Virtual Agents for Travel Assistance

Arūnas Miliauskas and Dalė Dzemydienė

Vilnius University, Institute of Data Science and Digital Technologies,
Akademijos st. 4, 04812 Vilnius, Lithuania
arunas.miliauskas@mii.vu.lt, dale.dzemydiene@mii.vu.lt

Abstract. The aim of this research is to propose a Belief-Desire-Intention (BDI) architecture-based approach for a virtual agent design. The presented case of a chatbot assistant in a travel domain demonstrates the necessity of the BDI architecture modification. The approach is taken for multiple BDI agent instances with a shared external knowledge base. The architecture was presented using Views and Beyond approach. A decision view was added to the architecture description to show alternatives and emphasize required modifications of the BDI architecture in the presented case.

Keywords: Software architecture, Belief-Desire-Intention architecture, Distributed databases, Travel assistance

1 Introduction

Our object of research is an assistant-broker agent that supports end-user in activities like travel planning, meeting arrangement. By using provided web services, the agent could complete actions behalf of the end-user.

The Foundation for Intelligent Physical Agents (FIPA) organization provided personal travel assistance specification [1]. It provides a scenario and an architecture of a potential distributed agent based system, where an end-user can get assistance and support in trip planning. That show quite big expectations towards intelligent agents for the semantic web. However, we still don't see today intelligent agents, like the personal travel assistant presented by FIPA, widespread.

BDI [2] is a well-know agent architecture. It is based on main concepts *Belief*, *Desire* and *Intention*, which ought to have an explicit representation in this type architecture. For us, one of most interesting aspects of the BDI architecture is an effect on modifiability. The architecture is based on interpreted methods, which are programmer created plans, that can be added to the running system.

In this research, our goal is to analyse BDI architecture application possibilities for building a virtual agent - travel assistant chatbot.

Lupeikiene A., Matulevičius R., Vasilecas O. (eds.):
Baltic DB&IS 2018 Joint Proceedings of the Conference Forum and Doctoral Consortium.

We use the framework [3] for design science. The result of a design is an artifact, based on actual or hypothetical stakeholder goals which are represented in Section 2.1.

The architecture is defined using the Views and Beyond approach [4]. It is arranged in views, where architecturally important structures are described from a particular viewpoint. We extend architecture descriptions with a decision view [5]. It contains architectural decisions and their relations, which are based on ontology of architectural decisions [6].

The contribution of this research is proposed extensions for the BDI architecture that overcomes identified limitations in travel planning assistant domain.

The remainder of this paper is structured as follows. Section 2 describes the architecture with context. An overview of related works is presented in 3 Final conclusions are drawn in Section 4

2 Assistant Agent Architecture

The architecture specification starts with business drivers presentation in Section 2.1. Then, based on [4], views are presented. We merge module decomposition view with context view in Section 2.2. It represents main structure of system and environment. Next view is Component and Connector (C&C) hybrid view, which describes a structure for one of main elements. This view is in Section 2.3. We add a decision view in Section 2.4 to demonstrate relations between architectural decisions and drivers. Main findings are discussed in Section 2.5

2.1 Business Drivers

Main drivers represents expected qualities, constraints and assumptions:

- *D1. Big amount of information.* We expect the system to be able collect and store data about potential flights, hotels, car hire services and their prices from service provider (SP) services and provide, apart precise date search, an exploration for best prices.
- *D2. Varying change rate for information.* Some information in travel domain is static and some changes often. For example we see frequent flight prices changes and stable flight schedule.
- *D3. Response latency* is taken from current SP services in the travel domain. We have observed that response latency usually is bigger than 1 second and less than 1 minute. The agent should have same response latency values.
- *D4. Chatbot.* This requires that assistant should be a chatbot.
- *D5. Non-critical availability*, since user could accomplish task "old way".
- *D6. Modifiability important.* This means emphasis on modifiability since adding new service providers may require a modification.
- *D7. No standartized services.* This assumption states that there should not be any expectations for standardized SP web services. The system must operate with existing services and this potentially requires an adaptation for each service provider.

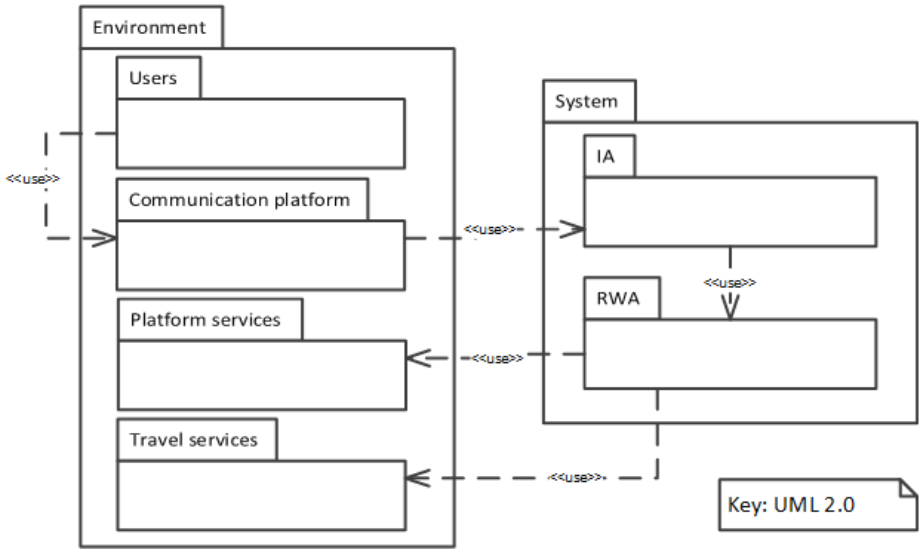

Fig. 1. Main modules and context

– *D8. SP uses REST.* This is an assumption that Representational state transfer (REST) architectural style [7] is used by SPs.

All drivers are used in Section 2.4 and visually represented in Fig. 3

2.2 Module Decomposition View with Context

Primary presentation. Diagram is presented in Fig. 1. It contains a context and main parts (internal structure).

Element catalog. System is envisioned to be placed in environment that consists of:

– *users*, which are end users that are expected to use system. However they don't use directly the system, but communicate using *communication platform*.
– *communication platform* is a medium that end-user uses for communication. It can be a platform like Skype, Facebook messenger, Slack and others.
– *platform services*, which are cloud based platform services that provide storage, computing services.
– *travel services*, which represents services providers like airlines, hotels etc.

The system is decomposed to:

– *IA* (stands for intelligent assistant) that encapsulates all intelligent behaviour,
– *RWA* (stands for real world adapter) - an intermediary between *IA* and SP web services.

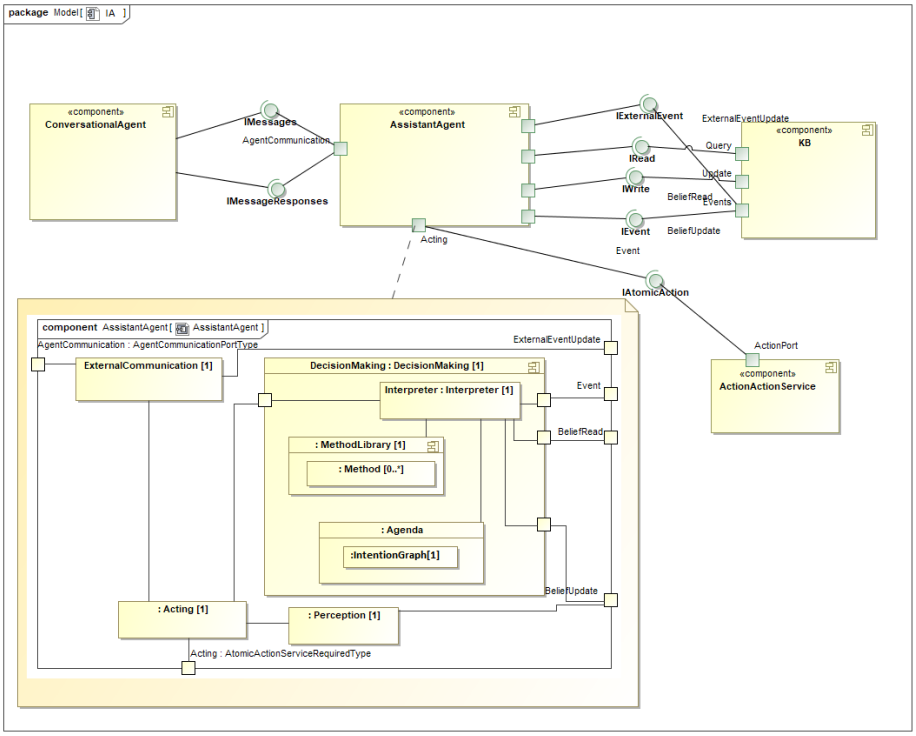

Fig. 2. IA module

Design rationale. The system decomposition is a result of the architectural decision *AD3. Decompose 2 RWA & IA* and is explained in Section 2.4. Another reason is that we want to encapsulate issues related to agent interoperation with environment into a separate module and then focus mainly on the intelligent (agent-based) part.

2.3 Hybrid C&C View and Module Decomposition View: IA

A C&C communicating processes view emphasizes concurrently running units and their interactions. Concurrently running units can be processes or threads. They interact by communication or synchronization and this interaction is represented using connectors.

Primary presentation. The primary presentation diagram is presented in Fig. 2. Here C&C style is mixed with module style. At highest level, the components are provided. However, an *AssistantAgent* component is further decomposed to details, where his internal structure is defined in a module view. The reason for this representation is desire to represent all BDI elements in single diagram.

Element catalog. *IA* module here is implemented by main 2 agent components:

- *ConversationalAgent*, which is responsible for maintaining conversation with end-user. It receives end-user messages identifies intentions and translates them to an agent communication language.
- *AssistantAgent*, which responsible for an assistance process. This component receives messages from *ConversationalAgent*, percepts environment and acts in it.

KB represents a knowledge base (KB) of the target system, which is implemented externally (from the *AssistantAgent* component perspective). *AtomicActionService* represent a component that implements an atomic action. Each atomic action is implemented by a separate component, which implements *IAtomicAction* interface. *Agent* is decomposed into modules:

- *ExternalCommunication* - module responsible for communication external entities. It exposes IMessages interface for receiving messages and uses provided IMessagesResponses to send responses.
- *Acting* module that is responsible for an intended action execution. It has link with *ExternalCommunication*, because sending a response message is also considered as act. It uses an IAtomicAction interface provided by *AtomicActionService*, to execute an action in the environment. Action results are sent to *Perception*.
- *Perception* module is responsible for sensing the environment. Since in the travel domain sensing is possible by querying services, which is treated as acting, this module receives data from *Acting* module.
- *DecisionMaking*, which further is decomposed:
  - *Interpreter*, which is responsible for sensing, choosing candidate methods and acting based on selected methods. It interprets statements in a method's body and executes them (acts).
  - *Agenda*, which represent agent active intentions (in BDI terms) in a *IntentionGraph*.
  - *MethodLibrary*, which represents agent desires (in BDI terms) and contains (in a method body) "recipies" how desires could be achieved.

Design rationale. Since all design decisions are defined in Section 2.4, here we relate the presented structure with these decisions. Components *ConversationalAgent* and *AssistantAgent* are introduced by the decision *AD5. Split conversation & assistance*. *AD7. BDI* defines structure of *AssistantAgent*. However, since this decision doesn't comply with requirements, other decisions are required, that shape the presented structure:

- *AD8. Shared KB agents* introduces a component *KB* outside *AssistantAgent* with defined interfaces.
- *AD13. Intention stack processing* introduces *IntentionGraph* and modifies *Interpreter* (changes not visible from this view).
- *AD14. Acting/sensing* changes are that *Perception* module is connected to environment only via *Acting* module.

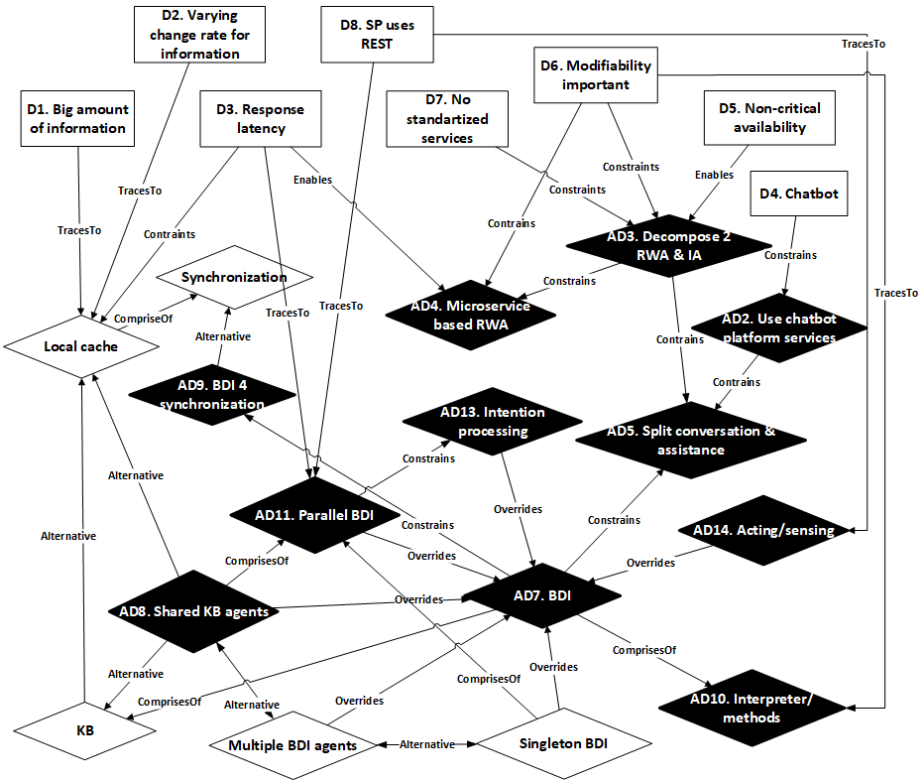

Fig. 3. Decision dependencies

2.4 Decision View

We present main architectural decisions and relations between them and business drivers.

Primary presentation. Primary presentation, in Fig. 3, consists of main requirements (represented by rectangles), architectural decisions (represented by diamond shapes) and their relations. It is simplified version leaving only decisions that are necessary for explanation and reasoning. Each decision is either in a decided or rejected/obsolesced state. Former ones are represented by filled shape and have an architecture decision identifier, whether latter ones don't have any identifiers and have corresponding shapes not filled.

Element catalog contains decisions:

– *Local cache* it means introducing data redundancy. It is an existence decision [6], stating that the element should appear is the system. However, it

is abstract and doesn't bring any implementation details. It is based on a business driver D3 for achieving required performance in context of other drivers D2, D1. This decision doesn't forbids, real-time access to SPs service for responding for end-user requests. Moreover, drivers D2, D1 suggest balancing between using cached information and real-time SP services access. Also this decision brings out a sub-decision to have data *synchronization*.

- *Synchronization* decision means that should be a mechanism for data synchronization, without explicitly stating how it is carried out.
- *AD2. Use chatbot platform services* is decision to use current cloud services providers as IBM Watson cloud, Microsoft Azure and others.
- *AD3. Decompose 2 WA & IA* represents decision splitting responsibilities into modules. It is intended to support modifiability requirement (D6). The other driver D5 enables this decisions. The decision is decomposed to sub-decisions that are related to constituent parts resulting in this split: *AD5. Split conversation and assistance* and *AD4. Microservice based RWA*.
- *AD4. Microservice based RWA* means that a module wraps SP services and exposes services to other modules as microservices.
- *AD5. Split conversation & assistance* is a decision to split responsibilities between assistance and conversation modules. Former module is responsible for whole assistance process and the latter one is for the conversation (recognizing end-user intents).
- *AD7. BDI* is the decision to use a BDI model for the assistance module. That brings out several decisions: *KB*, *AD10. Interpreter/methods*. However, it doesn't comply with performance requirements and modifications are needed. Additional decisions are introduced that override some properties of the BDI model.
- *KB* is an existence architectural decision introduced by *AD7. BDI* decision. This decision is alternative to the *Local cache* decision.
- *AD10. Interpreter/methods* is introduced by *AD7. BDI* decision. It defines that architecture should contain interpreter and library of methods, written in specific language.
- *AD8. Shared KB agents* is a decision to implement multiple agents based on a single shared KB. Therefore, it overrides the initial decision (*AD7. BDI*) and makes the *KB* decision obsolete (is alternative). We also describe some alternatives:
  - The use of singleton BDI agent responding multiple user requests - *Singleton BDI*. This requires significant modifications of the BDI agent. One of them is an introduction of concurrency. Another is a BDI processing cycle adaptation to handle multiple end-users requests in parallel. This alternative requires introducing parallel execution of actions *AD11. Parallel BDI*
  - Using multiple BDI agent instances where each has a separate KB - *Multiple BDI agents*. This requires an agent knowledge synchronization, sharing mechanism or multi-agent collaboration for single end-user request.

We see that described alternatives are more complex and requires more effort, than selected *AD8. Shared KB agents*.
- *AD9. BDI for synchronization*, by taking decision *AD7. BDI*, this enables use BDI agent for synchronization, what is an detailed alternative to abstract *Synchronization* decision.
- *AD11. Parallel BDI* is a decision to introduce the ability to execute simultaneously multiple information gathering (sensing) actions. The agent is expected to collect and aggregate many SPs information. We can not make assumption about a sufficient SP web service response latency. This leads to a decision querying these services in parallel and mixing results with the cached information. However, this property decision doesn't state how this is implemented.
- *AD13. Intention processing* refines the decision *AD11. Parallel BDI* with implementation details. It is about changes in BDI agenda part that intentions should be organized using a graph instead of a stack structure. The BDI processing cycle should also be modified to manage multiple active intentions in the graph.
- *AD14. Acting/sensing* is decision to relating acting and sensing. Since SP provide information using web services, sensing is a result of acting.

2.5 Results

The decision view demonstrates that the BDI architecture supports local cache and data synchronization capabilities. *Alternative* type relations between architecture decisions, captures that. However, modifications are needed. This is captured in the view with architectural decisions that *override* the main BDI introduction decision. The modifications are our selected decisions and main alternatives are captured in the decision view.

We demonstrate, that either modification, by enabling a capability of agent's action parallel execution, is required or there should be multiple agents serving single end-user request with a complex coordination mechanism. Latter approach is common in academia, but we select first, because we see it as less complex in provided context. This requires enabling agent with the single shared KB and parallel execution of actions.

3 Related Works

The research [8] reviews design choices for integrating agents with web services. Our approach corresponds to the presented theme 3. It is about agents composing simple (atomic) services. It is illustrated by integration web services with a Nuin BDI framework. However, the BDI architecture challenges pointed here by us are not covered in the referenced paper.

Most of research for integration agents with web services are based on a multi-agent system approach. The approach emphasizes collaboration of multiple agents, not on extensions of agent's internal architecture. Examples in travel

domain are [1], [9]. [10] used similar approach for integrated access to biological data sources.

Other approaches focused more on agent's internal structure. An example in travel domain is [11], where a prototype of smart tourist information points with Maxine platform based Embodied Conversational Agents (ECAs) is presented. A conversation with an end-user is governed by AIML (Artificial Intelligence Modeling Language) descriptions, which can be extended with custom scripts. These scripts can be used for calling external services. However, the architecture doesn't support building proactive agents. This is different from our and other BDI based approaches.

The are publications about integrating the Semantic Web and Agent programming. The most prominent paper is [12], which proposes JASDL (Jason AgentSpeak - DescriptionLogic). An AgentSpeak(L) implementation, Jason [13] is customized. However, the paper focuses on enabling ontological reasoning in an BDI agent.

4 Conclusions

In this paper we proposed an architecture for a chatbot assistant in a travel planning domain. The architecture was specified using Views and Beyond [4] approach. A decision view was added to the architecture specification to demonstrate alternatives, relate design decisions and trace them to business drivers.

The decision view shows two main advantages of a BDI architecture in this context. The first is a possibility to combine on-line SP service requests with cached data. The second is a support for the off-line data synchronization.

However the BDI architecture in this context requires modification. The main drivers are big amount information and performance requirements. We explore the alternatives and select an approach with multiple agents using a shared KB. Another required modification is enabling agent parallel execution of actions, which requires changing the representation of agent's intentions.

References

1. FIPA: Personal Travel Assistance Specification, 2000.
2. Anand S Rao and Michael P Georgeff. BDI agents: From theory to practice. In *Proceedings of the First International Conference on Multiagent Systems*, volume 95, pages 312–319, 1995.
3. Roel Wieringa. *Design science methodology*. Springer-Verlag Berlin Heidelberg, 1 edition, 2014.
4. Felix Bachmann, Len Bass, Paul Clements, David Garlan, James Ivers, M. Little, Paulo Merson, Robert Nord, and Judith Stafford. *Documenting Software Architectures: Views and Beyond*. Addison-Wesley Professional, second edition, 2010.
5. Philippe Kruchten, Rafael Capilla, and Juan C. Dueñas. The Decision View's Role in Software Architecture Practice. *IEEE Software*, 2009.

6. Philippe Kruchten. An Ontology of Architectural Design Decisions in Software-Intensive Systems. In *2nd Groningen workshop on software variability*, pages 54–61, 2004.
7. Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. Doctoral dissertation, University of California, Irvine, 2000.
8. Ian Dickinson and Michael Wooldridge. Agents are not (just) web services: considering BDI agents and web services. In *Proceedings of the 2005 Workshop on Service-Oriented Computing and Agent-Based Engineering (SOCABE'2005)*, Utrecht, The Netherlands, 2005.
9. Dickson K.W. Chiu, Yves T.F. Yueh, Ho Fung Leung, and Patrick C.K. Hung. Towards ubiquitous tourist service coordination and process integration: A collaborative travel agent system architecture with semantic web services. *Information Systems Frontiers*, 11(3):241–256, 2009.
10. Francisco García-Sánchez, Jesualdo Tomás Fernández-Breis, Rafael Valencia-García, Juan Miguel Gómez, and Rodrigo Martínez-Béjar. Combining Semantic Web technologies with Multi-Agent Systems for integrated access to biological resources. *Journal of Biomedical Informatics*, 41(5):848–859, 2008.
11. Piedad Garrido, Javier Barrachina, Francisco Martinez, and Francisco Seron. Smart tourist information points by combining agents, semantics and AI techniques. *Computer Science and Information Systems*, 14(1):1–23, 2017.
12. Thomas Klapiscak and Rafael H. Bordini. Jasdl: A practical programming approach combining agent and semantic web technologies. In Matteo Baldoni, Tran Cao Son, M. Birna van Riemsdijk, and Michael Winikoff, editors, *Declarative Agent Languages and Technologies VI: 6th International Workshop, DALT 2008, Estoril, Portugal, May 12, 2008, Revised Selected and Invited Papers*, pages 91–110, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.
13. Rafael H. Bordini, Jomi Fred Hübner, and Michael Wooldridge. *Programming Multi-Agent Systems in AgentSpeak using Jason*. 2007.