Business and Model-Driven Development of BDI Multi-Agent Systems

Article *in* Neurocomputing · December 2015

DOI: 10.1016/j.neucom.2015.12.022

2 authors:

Yves Wautelet
KU Leuven
123 PUBLICATIONS **1,053** CITATIONS

SEE PROFILE

Manuel Kolp
Université Catholique de Louvain - UCLouvain
208 PUBLICATIONS **3,231** CITATIONS

SEE PROFILE

Neurocomputing ∎ (∎∎∎∎) ∎∎∎–∎∎∎

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Business and model-driven development of BDI multi-agent systems

Yves Wautelet [a,*], Manuel Kolp [b]

[a] KU Leuven, Belgium
[b] Université catholique de Louvain, Belgium

ARTICLE INFO

Article history:
Received 19 February 2015
Received in revised form
29 August 2015
Accepted 6 December 2015
Communicated by Ngoc Thanh Nguyen

Keywords:
Service modeling
BDI software system
I*
Actor responsibility assignment
Multi-agent system
Model-driven engineering
Agent-based development

ABSTRACT

Model-driven development allows IT professionals to specify the system functionality, organization and behavior in a logical or platform-independent manner. Modeling using services allows domain analysts to focus on the added-value and core business the enterprise offers to its stakeholders. Those services are coarse-grained elements able to encapsulate a composition of business process models. The framework presented in this paper provides models together at strategic, tactical and operational levels to develop an agent-oriented software system. The strategic level is concerned with long-term decisions; this top-level uses a service model to understand the business' high-level (added) values as well as the Quality Expectations and the threats they face. The tactical level is concerned with a broader description of the business processes automated by the system; the i* strategic dependency and rationale models are used here to further document the service behavior. Actors' accountability and responsibility can be determined in the visual representation of these strategic and tactical levels. Finally, i* models are mapped into a set of operational models to document the (multi-agent) system behavior when achieving modeled functionalities. These operational models instantiate the Belief/Desire/Intentions (BDI) paradigm proposing entities – the agents – mapping as closely as possible the real life organization. The paper thus builds a business-driven transformation process leading to a run-time agent-architecture in a single and common framework. It both uses existing models and introduces or refines existing ones to dispose of a method ensuring better alignment and traceability between the business and the IT system.

1. Introduction

1.1. Research context

Many modern software development methodologies are said to be *Model-Driven* in the sense that the whole development process can be traced from or driven by high-level modeled entities. For instance, object-oriented development methodologies such as the *Unified Process* inspired ones (*RUP, OpenUP, EUP, AUP, …* [1–4]) are said to be *use case driven*, meaning that the entire process is driven by the system functionality and behavior identified as use cases at the requirement analysis and/or business modeling stage. Similarly, implementation methodologies for ERP and e-business systems [5] such as *Accelerated SAP* (*ASAP*), *Fast Track*, *Business Integrated Methodology* (*BIM*) or *Sure Steps* or in some case even *PRINCE 2* [6] may be considered business processes driven in the sense that the life cycle is, in this case, driven by business functionality and activity identified during the business (process) modeling step.

Besides, in model-driven development, highest level analysis elements are called *scope elements* and are consequently useful not only to share a common high-level vision with stakeholders, but also to estimate the project effort on a non-redundant basis, for evaluating related risks and opportunities brought by software adoption, etc.

Defining adequate scope elements is a key factor for a successful adoption of a software development methodology. Indeed, such element granularity must be adequate and the focus on a core functionality of the application is crucial to determine one particular aspect of the software to build. Agent-oriented development methodologies such as Tropos [7–11] have proposed various concepts to represent and develop software systems; some are coarse-grained (e.g. goals, tasks) and other fine-grained (e.g. beliefs, desires, intentions). Nevertheless, agent and requirement-driven methodologies such as Tropos still lack to adopt a clear "red-threat" from the strategic to the operational levels with a direct impact on scalability (see [12]).

1.2. Contributions

This paper is an effort to propose a clear model-driven framework to develop agent-oriented software proposing strategic, tactical and operational views. To this extent, it addresses the lacks and deficiencies of classical i* (i-star, see [13–16]) models to furnish adequate scope elements. For this purpose, it refines a proposal

* Corresponding author.
E-mail address: yves.wautelet@kuleuven.be (Y. Wautelet).

http://dx.doi.org/10.1016/j.neucom.2015.12.022
0925-2312/

from [12] to define a strategic analysis model driven by the concept of *Service*. It genuinely introduces elements of quality and risk management at this strategic level and formalizes the proposal. It also proposes to study the actors' responsibility assignment within the strategic and tactical perspectives. Classical i* models are not left out but used for tactical knowledge representation. Finally, the Multi-Agent System (MAS) design – constituting the operational perspective – is represented through three different models introduced in the paper. These are aimed to implement agent software with the cognitive *Belief*, *Desires*, *Intentions* (*BDI*) paradigm in mind (see [17–20]), a simple but efficient reasoning model that allows us to capture human rationale. We thus address the representation of the system-to-be. The implementation of the operational models is nevertheless outside the scope of this paper, but an implementation model for the proposed MAS design has been covered in previous work (see [21]).

In short, the paper formalizes a strategic model for knowledge representation as well as design diagrams following the BDI paradigm; the tactical middle layer constituted by the i* strategic dependency and strategic rationale diagrams is left as-is. This gives a business and model-driven perspective for Tropos allowing full traceability of elements from strategic to operational levels.

1.3. Paper structure

The paper is organized as follows: Section 2 overviews related work and positions the present proposal. Section 3 motivates the need for a high-level vision. Section 4 defines a model-driven framework for business modeling based on services while Section 5 formalizes our service-driven agent modeling approach. Section 6 specifically highlights the transformation process. The transformation process is illustrated on a case study in Section 7. Finally, Section 8 concludes the paper.

2. Related work

Most Agent-Oriented Software Engineering (AOSE) methods focus on the design stage of a MAS and poorly take organizational analysis as an important step into the software development. Furthermore, some AOSE methods claiming to be requirements-driven only rely on UML use-cases as development scenarios (like for example the *Multiagent System Engineering* (*MaSE*) method [22]); Tropos uses advanced organizational analysis through i* for business process analysis as a first step in the agent-software development. Nevertheless, the i*-driven approach from Tropos has some drawbacks (see Section 5.1.1) notably to represent a strategic perspective offering enough scalability abilities for dealing with large projects and to clearly forward engineer elements into a MAS design. With respect to the models included in the Tropos process as presented in [7–11], our framework:

- Includes the *Strategic Services Model* (*SSM*) which through its constituting elements allows to drive the model transformation process; more particularly it allows to:
  - deal with the issues of classical i*/Tropos notably highlighted by [12]. In our case, we are particularly interested by the way it offers to deal with scalability issues. Indeed, huge projects induce huge sets of i* elements (notably goals and tasks) and rapidly become unmanageable. Managing the project on the basis of services (i) addresses this issue particularly well (see [12,23]) and (ii) allows us to develop a software application highly aligned with the business the company is exercising;
  - tackle quality management and risk management basics at a high level of abstraction. Indeed, services can be impacted by *Quality Expectations* and *Threats* the overall company has/is facing with respect to its IT strategy. These are thus identified at strategic level and later forward engineered at tactical level into a set of i* softgoals, goals and/or tasks;
  - allows us to study involved actors' accountability – service governance perspective – and responsibility – service management perspective – in a unified framework.
  - manage the software project and deal with planning issues. The present paper focuses on the transformation issues related to model driven development and this element is thus outside the scope of the paper but it can also be used in an iterative project management perspective as illustrated in [24].
- Includes design models to forward engineer i* models into a BDI Agent-Oriented Design. The implementation of these models is outside the scope of the paper but their implementation using the *Java Agent DEvelopment Framework* (*JADE*) [25] can be found in [21]. JADE is a framework used for implementing MAS which conforms to the FIPA standard (see [26]). JADE simplifies the MAS development while ensuring standard compliance through a comprehensive set of system functions and their related agents.

Next to Tropos, other AOSE methods have been proposed; we position in the rest of this section our contribution with respect to these. Gascueña et al. [27] study model-driven techniques for the development of MAS. It notably compares, on the basis of a set of features, the technological aspects of INGENIAS [28], Prometheus [29] and PIM4Agents [30,31]; it then further studies Prometheus. When compared to Prometheus, our framework offers a strategic and tactical layer to drive the software process. Prometheus indeed directly starts the development with basic system goals and functionalities developed in the form of use-case scenarios. Moreover, Prometheus targets the *JACK intelligent agents* [32] as development platform. Our framework remains independent of any implementation language even if an implementation guidance with JADE is provided in [21]. The framework developed in this paper is business-driven thus tackling a layer that has not been linked to agent-design concepts through a transformation process in previous work.

To the best of our knowledge, no other framework or MAS development method has furnished a complete and consistent solution. For instance, *Multi-Agent Systems Development Methodology* (*MASD*) [33] claims to address the whole life cycle of an AOSE development; it nevertheless only envisages requirements as defined scenarios issued of use-cases.

Finally, we also highlight that Descartes Architect [34] is a CASE-Tool has been developed to support the creation and edition of the diagrams of our framework.

3. The need for a high level vision

Management and organizational theories involve several layers for decision making. Indeed, decisions do not have the same impact – from a marginal short term consequence to a major long-term strategy – so that their time horizon is variable. Traditionally, management sciences identify three levels of decision making in order to differentiate time horizons and resources that should be allocated:

- The Strategic Level in which decisions are top-level non-structured knowledge processes concerning general direction, long-term goals, philosophy and values of the organization.
- The Tactical Level in which more concrete, semi-structured decisions are taken aiming at implementing the strategy defined at the corporate level. The business units adapt this strategy in terms of policies under which the operations will take place.
- The Operational Level in which daily structured decisions are made to support tactical ones. Their impact is immediate, on a short-term, short range, and usually low cost. Operational

Please cite this article as: Y. Wautelet, M. Kolp, Business and model-driven development of BDI multi-agent systems, Neurocomputing (2016), http://dx.doi.org/10.1016/j.neucom.2015.12.022

decisions can be pre-programmed, pre-made, or set out clearly in policy and operation manuals.

Similarly, software development methodologies are often divided into a set of stages (called for example phases into Tropos and disciplines in the Unified Process). The analysis stage consists of different complementary aspects to represent the problem to solve at various levels of abstraction using various conceptual or logical views. The different decision levels are embodied by individuals having different visions and expectations on the software product in progress. Indeed, while the software application is mainly used by operators (at the operational level), requirements and needs are often supervised by the middle management (tactical level) while the top management only has to perceive the main functionality as well as the business long-term implications of a software package (strategic level).

4. Models perspective: from business models to BDI multi-agent systems

This section positions the different models that are used in the development framework. Indeed, each model has a specific purpose and is located at a defined abstraction level. Models are intended to be complementary.

4.1. Models for addressing each decision level of an AOSE development

Fig. 1 presents the models included in our framework through the decision pyramid.

The analysis level are:

- The strategic level is tackled through the *Strategic Services Model*[1] described and formalized in Section 5.1. This level provides a coarse grained representation of the software application through its main services. Actors' accountability – who is (legally) in charge of ensuring the proper fulfillment of the service for which other actor – is also represented and determined here. Environmental factors influencing positively or negatively the provided services are also modeled at this stage (see Section 5.1). This model can be used as a reference vision document for stakeholders as well as a guidance for quality management, risk management, software process management (*SPM*, more precisely iteration planning and effort planning). SPM is outside the scope of the present paper but is partly documented in [24].
- The tactical level is materialized through *i* Strategic Dependency* and *Strategic Rationale Models* (see [13–16]). This level provides a finer grained description of the softgoals, goals, tasks and resources involved in the achievement of all the services. Actors' responsibility – who concretely does what – is also represented and determined here.

The models developed in the analysis level are then mapped into a series of design models representing, through different points of view, the way agents can behave at run-time. They consist of:

- The *Architectural Model* that operationalizes the analysis models into an agent-oriented architecture using concepts such as *beliefs*, *events*, *plans*, and their relationships.
- The *Communicational Model* that specifies the temporal exchange of events between agents.
- The *Dynamic Model* that captures the synchronization mechanisms between events and plans.

4.2. Models complementarity

The models proposed in our approach are complementary in the sense that each of them regrouped on different level offer a 360° perspective comparable to the 4+1 architectural view model proposed for object-oriented systems in [35].

The *Strategic Service Model* allows us to represent, in an aggregate and non-redundant manner, the main services the application has to offer.

The *Strategic Dependency Model* depicts the service in terms of middle grained elements using goals, tasks (i.e., business processes) and also resources. It answers the question: "*What is the service about?*".

The *Strategic Rationale Model* fulfills this task by identifying and giving the definition of *capabilities* (tasks and goals of each agent) that it can use to realize its goals. However, at this level, the definition of capabilities is completely independent from the way in which they will be implemented. This allows the designer to consider alternate manners to program and deploy these capabilities.

The last three models, at the operational level, give more specific information related to the design of the service implementation.

The *Architectural Model* describes "*How can each capability and agent be implemented?*". Its role for designing agent-oriented systems corresponds to the role of the class, object and composite structure and component model for designing object-oriented systems.

The *Communication Model* describes "*How do agents exchange information between them?*". Its role into the design of agent-oriented systems is similar to the role of the sequence and collaboration/communication diagrams for designing object-oriented systems.

The *Dynamic Model* describes "*How is the flow of control between capabilities modeled?*". Its role corresponds to the role of the statechart/activity diagrams for object-oriented systems.

Since we use only six models, the question of models completeness may raise as it might be the case for other software development frameworks. A common answer to this question is that "*the design process is finalized when all the necessary information about the system to be constructed has been described and the programmers could use this description to construct the system without any need for further descriptions*". In our framework, the

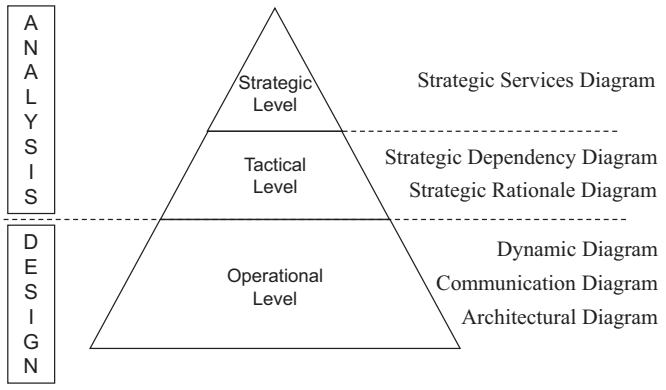

Fig. 1. The Software Development Models: A layered view.

[1] Note that we refer to a *Model* when we refer to the meta-model depicting the elements of the conceptualization and to a *Diagram* when we refer to its instantiation on a case study. The same distinction is made for the other models of the framework.

Please cite this article as: Y. Wautelet, M. Kolp, Business and model-driven development of BDI multi-agent systems, Neurocomputing (2016), http://dx.doi.org/10.1016/j.neucom.2015.12.022

design process through these six models gives us enough information to describe the characteristic of an agent-oriented system.

5. Models and elements description

This section presents the genuine models of the framework, i.e. the SSM, the Architectural Model, the Communication Model and the Dynamic Model. A meta-model of the i* Strategic Dependency and Rationale Models we rely onto can be found in [36]. The section studies the elements defined within these models in detail. The transformation process between these elements will be the focus of Section 6.

5.1. Strategic Services Model: concepts and notions

The meta-model of the SSM is presented in Fig. 2; it refines the proposition of [12] by providing a view allowing to distinguish scope elements – the *services* – encapsulating sets of i* process elements facing Quality Expectations (QE) and threats. The instance of the SSM will, in the development process, be used to define the development in terms of coarse-grained elements. The elements of the SSM are instantiated to provide a *Strategic Service Diagram* (*SSD*). This section offers a (genuine) full conceptualization of our refined proposal.

5.1.1. Services

The i* framework can be evaluated on a series of nine features following [12]: *refinement*, *modularity*, *repeatability*, *complexity management*, *expressiveness*, *traceability*, *reusability*, *scalability* and *domain applicability*. Those features are exhaustively assessed on the basis of a *not supported*/*not well supported*/*well supported* scale. Notably they enlighten what is clearly needed to extend the i* framework with mechanisms to manage granularity and refinement. Indeed, Pastor et al. [12] point out the lacks of mechanisms in i* for defining *granules* of information at different abstraction levels to structure, hierarchies or aggregate the semantics represented in the model. One of the flaws of i* is actually that all of the organizational modeling elements are represented on a unique abstraction level with poor hierarchy and composition/aggregation. Moreover, except for specifying abstract primitives as building blocks, analysts must be provided with guidelines to model a complete business setting through a set of organizational processes. These building entities could then be enriched into a set of more specific components that capture a certain organizational behavior. An high-level business view is then required such as the *Business Service Architecture* proposed in [12] proposing a software analysis focusing its activity on the *values* the Enterprise offers to the Customers. Those values are called *Services* and used as basic granules of information that encapsulate a set of i* business process models. The services the enterprise offers are typically used as high-level scope elements while business processes fulfilling those services are then decomposed and refined. This approach allows us to combine the intentional and social characteristics of i* with the "traditional" business process modeling and evaluate the most suitable development strategy.

In [37,12] business services are described as "*an abstract set of functionalities that are provided by a specific actor*" while "*an actor can be an organizational entity … that uses or offers services*". This approach is here extended through the definition of the SSM.

Multiple definitions of the service concept can be found in the literature with notably an impact on the evaluation of their granularity. In the context of this paper, we rely to the conclusions of [38] and define services as "high-level" elements i.e. *coarse-grained granules of information that encapsulate an entire or a set of processes*. This view is in accordance with the one of [39] which proposes an ontology for their proper representation. Within their conceptualization they notably distinguish the *Service Commitment* – prescriptive level – and the *Service Process* – design and implementation levels – useful in the transformation approach covered into the rest of this paper.

We define $\langle s_j^i, pm_j \rangle$ as a service s_j, where s_j^i provides the details of the specification (in some language) of the service and pm_j is the process model of the service expressed here using the i* elements.

5.1.2. Actors

Actors are intentional entities used to model people, organizations or software systems that are performers of some actions.

The *Role* concept inherits from *Actor* and, in the context of service modeling, it can consider *Service Provider* (*SP*) or *Service Consumer* (*SC*) as instances. The *Dependency* class materializes the dependency relationship between the service consumer and the service provider in the context of a service commitment (prescriptive level). The service consumer is the depender actor while the service provider is the dependee. More formally:

An actor a_j can be defined as a tuple $\langle\{(rol_i, q_{rol_i}), \ldots, (rol_{i+m}, q_{rol_{i+m}})\}, Act\rangle_j$, where rol_i is a role the actor enacts to fulfill

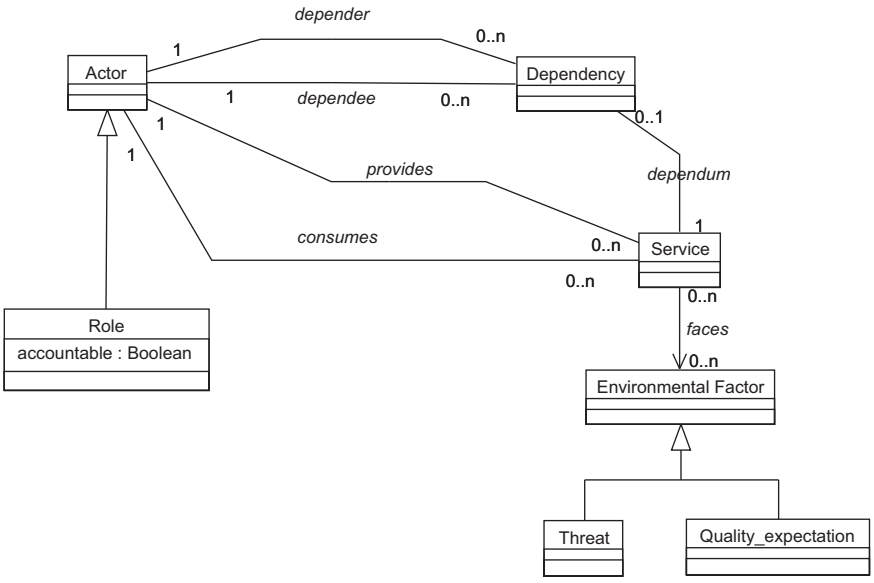

Fig. 2. Strategic Services Model: A meta-model.

Please cite this article as: Y. Wautelet, M. Kolp, Business and model-driven development of BDI multi-agent systems, Neurocomputing (2016), http://dx.doi.org/10.1016/j.neucom.2015.12.022

or consume services at quality level and cost q_{rol_i}. Played roles form a set *AR* of pairs, namely quality level and cost with values. *Act* is assumed to contain all additional properties of the actor necessary for its definition.

Format and content of *Act* will depend on the structure of the application domain. In practice, an actor usually plays several roles.

In a more formal way we can say that: $\forall a_j \in A, \exists AR^{a_j} \subseteq R \times Q$ with $AR^{a_j} \neq \varnothing$ so that an actor role is defined as $a_k^{AR^{a_j}} \in AR^{a_j}$ and $a_k^{AR^{a_j}} = \langle a_j, rol_k, q_{rol_k} \rangle \ \forall k = i..m$ where A is the set of actors a_j and AR^{a_j} is the set of roles ($rol_k \in R$) that the actor a_j can enact along with their quality levels ($q_{rol_k} \in Q$).

It is from primary importance to understand that at this highest (strategic) level the dependency relationship materializes the *accountability* of the SP (or dependee) with respect to the delivery of the Service (which is the dependum) to the SC which is the depender. Indeed, in the perspective of the responsibility assignment matrix, the accountability of the SP means that he must ensure service delivery within the agreed quality levels even if he is not *responsible* for performing each action (this can be evaluated in the tactical view) taken in the context of this service delivery. In other words, if something goes wrong the SP is the actor to blame meaning that he is in charge to provide a solution or compensation to the SC. This highest-level can thus be said of track the service governance while the tactical level will track its management.

5.1.3. Environmental factors

A service fulfillment is influenced by *Environmental Factors*; indeed, QE can potentially reinforce competitive advantage of service fulfillment while threats can prevent taking benefits from the added-value furnished by the service. Since the discussion takes place at the coarse-grained level, elements are expressed in an aggregated manner in order to evaluate the alignment between the IT supported services and the organization's long-terms positioning. A complementary ontological-basis can be used to qualify and quantify the QE and threats on a case by case basis. The purpose here is to express this in a generic way to illustrate how the software process is managed.

QE must firstly be distinguished from traditional softgoals in the sense defined by [40,7,41,42]. Indeed, following [40,7], "*a soft-goal is a condition or state of affairs in the world that the actor would like to achieve. But unlike a hard-goal, there are no clear-cut criteria for whether the condition is achieved, and it is up to the developer to judge whether a particular state of affairs in fact achieves sufficiently the stated soft-goal*" while [42] enlights that "*soft-goals prescribe preferences among alternative system behaviors*". Since we address a higher level business view of the system's services, we need an abstraction where we can specify the "quality" concerns of the organization in line with its long-term strategy. In other words, QE are the stakeholders' desires to align the system-to-be with the competitive positioning defined for the long run. Softgoals refer to the actor-level concerns the system-to-be should have to cope with. In this sense, softgoals describe conditions, states or preferences of a system-to-be while QE are aimed to enhance the value added for the organization implementing the software system. Within our modeling approach, a QE is expressed and quantified in the form of a constraint/concern onto a particular service through a degree of excellence (this is used for managing the whole software process and not within the transformation process so it is not documented here since it is outside the scope of the paper; it is nevertheless documented in [24]). A QE is later refined into a series of i* softgoals, goals and Tasks with respect to the transformation process (see Section 6).

A Threat describes an event that can negatively affect the proper fulfillment of a service or that can be the result of the misuse of a service process both in terms of achievement and degradation of quality. The Threat is expressed as an aggregated risk with a quantification of the negative impact and a likelihood of occurrence (this is used for managing the whole software process and not within the transformation process so it is not documented here since it is outside the scope of the paper; it is nevertheless documented in [24]). A Threat is later refined into a series of i* softgoals, goals and tasks with respect to the transformation process (see Section 6).

5.2. Architectural Model: concepts and notions

This section introduces the internal architecture of software agents; their relationship with the tactical level is discussed in Section 6.

5.2.1. Meta-model

The Architectural Model introduces agent-oriented implementation concepts such as *agents*, *beliefs*, *events* and *plans*. This dimension assumes the role of the logical view in an object-oriented approach. However, agent-oriented systems introduce other concepts, we then need an extended way to model them.

As mentioned earlier, our architectural and implementation target is specifically the BDI model, a cognitive agent model whose main concepts are *Beliefs*, *Desires* and *Intentions*; for full specification of these concepts the interested reader is invited to refer to [18].

An agent realizes the capability (i.e., a goal or task) identified at Analysis level using its *plans*. A plan is composed of a sequence of actions that an agent takes. A plan is then considered as a capability materialized or, in other words, capabilities are operationalized as plans.

The knowledge that an agent has (about itself or its environment) is stored in its *beliefs*. An agent acts in response to the *events* that it handles through its plans. A plan, in turn, is used by the agent to read or modify its beliefs, and send events to other agents or post events to itself.

Fig. 3 depicts a meta-model of the concepts and their relationships to build the architectural dimension. Each concept presented in this figure is studied in more detail in the following sub-sections.

5.2.2. Agent structure

Fig. 4 proposes a generic template for the *Agent* concept. The definition of an agent is composed of five parts: *Attributes*, *Events*, *Plans*, *Beliefs* and *Methods*.

We typically refer in this work to BDI agent implementation platforms such as JADE, JACK Intelligent Agents or the Jadex BDI Agent System [43,44,20].

The agent class allows to specify:

- the declaration of agent attributes;
- the events (both internal and external) that the agent handles; can post internally (to be handled by other plans); can send externally to other agents;
- the plans that the agent can execute;
- the beliefs the agent can use and refer to. The beliefs of an agent can be of type *private*, *protected*, or *public*. A *private* access is restricted to the agent to which the belief belongs. *Protected*

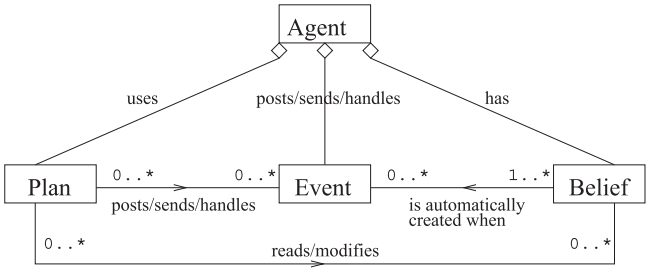

Fig. 3. Concepts of the BDI architectural dimension.

Please cite this article as: Y. Wautelet, M. Kolp, Business and model-driven development of BDI multi-agent systems, Neurocomputing (2016), http://dx.doi.org/10.1016/j.neucom.2015.12.022

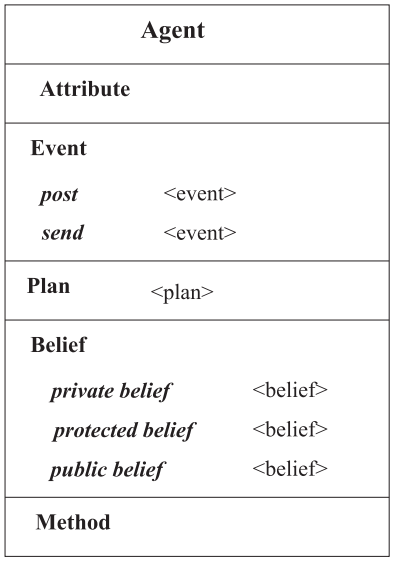

Fig. 4. Agent template.

access is shared with other agents of the same class, while *public* access is unrestricted;
- the declaration of agent methods (e.g., the constructor of an agent).

The details about the structure of the *Event* can be found in Appendix A and the ones about the structure of the *Plan* can be found in Appendix B.

5.3. Communication Model: concepts and notions

Agents interact with each other by exchanging events. The Communication Model introduces, in a temporal manner, events exchanged in the system. We adopt the sequence diagram model proposed in AUML [45,46] and adapt it with some extensions for the concepts that are introduced in the architectural dimension. These extensions allow us to model the events exchange between agents.

Fig. 5 depicts basic elements for agent communication. Rectangles express individual agents or sets (i.e., roles) of agents. For example, an individual agent could be labeled `David/Client`. Here `David` is an instance of agent playing the role of `Client`. `David` could also play the role of `Service Provider`, `Employee`, etc. The basic format for the box label is `AgentName/Role`. Therefore, we could express the various situations for `David`, such as `David/Client` (agent `David` plays the `client` role) and `David/Employee` (agent `David` plays the `employee` role). The rectangular box can also indicate a general set of agents playing a specific role. Here, just the word `/Client` or `/Employee` would appear. The `AgentName/Role` syntax is already part of AUML. Fig. 5 extends AUML by labeling the arrowed line with an event, instead of an object-oriented style message.

In order to keep the focus on the elements of the model and to avoid too heavy descriptions, further description on the Communication Model notably involving concurrent communication has been put in Appendix C.

5.4. Dynamic Model: concepts and notions

As discussed earlier, a plan can be invoked by an event that it handles and can create new events. Relationships between plans and events can rapidly become complex. To cope with this problem, we propose to model the synchronization and the

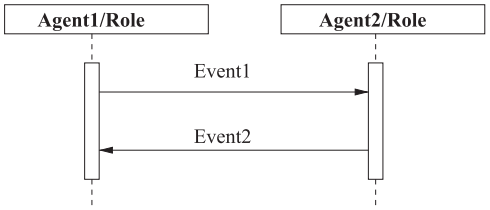

Fig. 5. Basic format for the communication dimension.

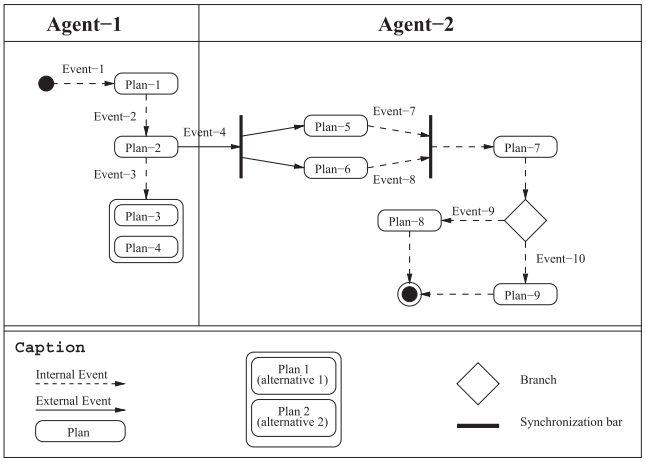

Fig. 6. Dynamic diagram template.

relationships between plans and events with activity diagrams extended for agent-oriented systems. These diagrams specify the events that are created in parallel, the conditions under which events are created, which plans handle which events, and so on.

Fig. 6 shows a template for dynamic diagrams. Each agent constitutes a swimlane of the diagram (e.g., `agent-1` and `agent-2`). The plan (e.g., `Plan-1`) is represented in a round-corner box and is placed in the swimlane corresponding to the agent that it belongs to. An internal event (e.g., `Event-2`) is represented by a dashed arrow and an external event (e.g., `Event-4`) by a solid arrow. A BDI event may be handled by alternative plans (e.g., `Plan-3` and `Plan-4`) that are enclosed in a round-corner box. Synchronization and branching are represented as usual.

At a lower level, each plan could also be modeled by an activity diagram for further detail if necessary.

6. Transformation process: from coarse-grained to fine-grained elements

Fig. 7 graphically illustrates the whole transformation process of the presented framework. The steps will be illustrated and commented on a case study in Section 7. The arrows crossing the levels show the transformation (forward engineering) of elements.

Table 1 depicts each of the transformation steps of the process.

7. Validation: application of the transformation process on a case study

The transformation process of the framework proposed in this paper is illustrated on the development of a collaborative platform for *travel management*. Roughly speaking, the whole system provides a common applicative package to all actors involved in travel organization from service providers (i.e., hotel infrastructure managers, long haul transportation companies and local transportation companies) to final

Please cite this article as: Y. Wautelet, M. Kolp, Business and model-driven development of BDI multi-agent systems, Neurocomputing (2016), http://dx.doi.org/10.1016/j.neucom.2015.12.022

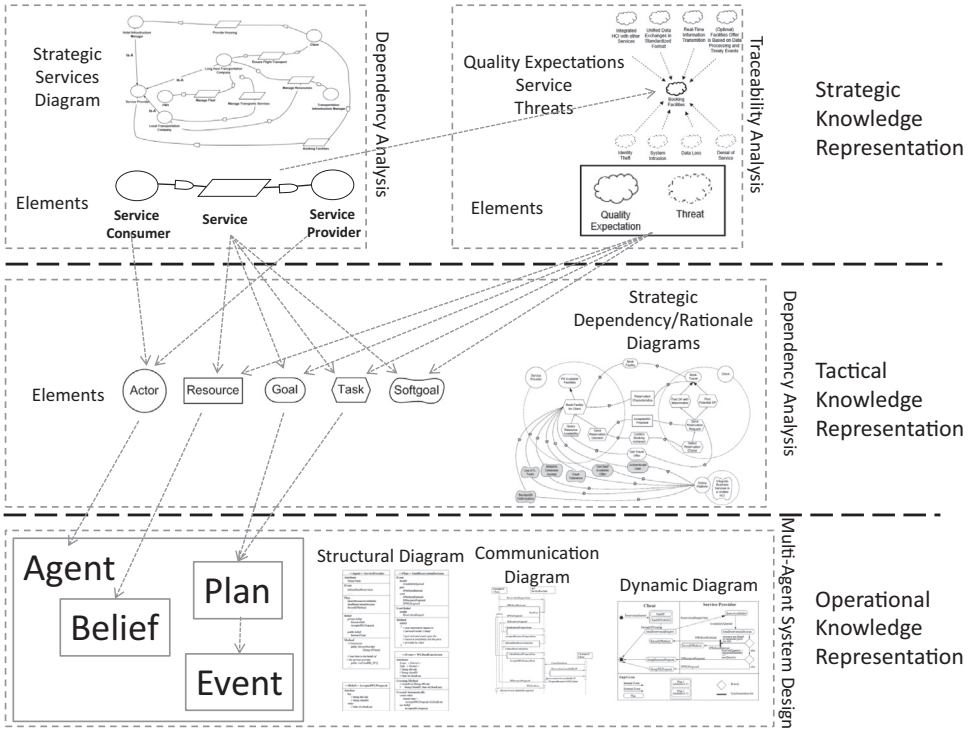

Fig. 7. Transformation process.

clients. Indeed, the travel supply chain involves series of actors incarnated by various collaborating or competing companies where several roles played by lots of individuals are interacting to achieve common as well as individual goals. In such a context, one needs powerful tools to represent travel flows, to deliver information to supply chain partners and to ensure taking all the possible advantages of e-business.

7.1. Strategic level

7.1.1. Services

At the left side of Fig. 8, the SSD for the travel planning collaborative platform is depicted; it provides users with a package of services and actions designed to encourage efficient, safe, healthy and sustainable traveling (including transport, accommodation, etc.) options.

As evoked, this strategic level depicts the services that the collaborative platform must offer in terms of coarse-grained elements. Those services are *Booking Facilities*, *Manage Resources*, *Manage Transports Services*, *Ensure Flight Transport*, *Provide Housing* and *Manage Fleet*. Due to a lack of space we will only focus on the development of the *Booking Facilities* service. That is why, on the left part of Fig. 8, the study of the QE and threats is performed for this service only.

7.1.2. Actors, their accountability and responsibility

In the *Booking Facilities* service, the *Client* depends on a SP[2] for booking facilities like accommodation or a flight; the *Client* plays the role of *SC*. This also means that the SP is in charge of ensuring the achievement (because he is accountable for it) of all the process elements related to booking a facility for the SC; this set is then documented/refined at tactical level (see Section 7.2) where the responsibilities of each element can be evaluated.

7.1.3. Environmental factors

At this strategic level, QE and threats concerning these coarse-grained scope elements must be identified to follow them up in the forward-engineering activities. The right side of Fig. 8 shows these Environmental Factors for the *Booking Facilities* service of the collaborative platform to be developed. As mentioned in the legend, the upper side of the figure shows the QE. Those QE are *Integrated HCI with other Services*, *Unified Data Exchanges in Standardized Format*, *Real-Time Information Transmission* and *(Optimal) Facilities Offer is Based on Data Processing and Timely Events*. Similarly, the threats are *Identity Theft*, *System Intrusion*, *Data Loss* and *Denial of Service*.

The SDD as well as related QE and Threat elements constitute the strategic layer of the to-be software platform.

7.2. From strategic to tactical level: towards an i* diagram

7.2.1. Service transformation

The transformation process starts from the strategic level where the software problem is represented into a set of services required by *SC* and provided by *SP*. The set of processes relating to the service fulfillment are depicted at tactical level in the form of i* goals, softgoals, tasks and resources. The *SC* and *SP* are forwarded into i* actors at tactical level.

The SRD for the *Booking Facilities* service of our case study is depicted in Fig. 9. The `Online Platform` is intended to support the `Client` and `Service Provider` actors in their activities related to the fulfillment of the service.

The `Service Provider` actor wants to *Fill Available Facilities* and, to this end, he achieves the Task *Book Facility for Client*. This Task is then refined into *Query Resource Availability* and *Send Reservation Decision*.

The `Client` actor depends of the `Service Provider` actor to fulfill the *Book Facility* goal and to *Confirm Booking Achieved* task. The *Service Provider* depends on the `Client` actor for the resources *ReservationCharacteristics* and *AcceptedWLProposal* (WL stands for Waiting List).

[2] Note that, in a SSD, the dependee actor is stereotyped *SP* and the depender *SC* but here Service Provider refers here (for once) both to the stereotype of the actor and also to its name. In the rest of this paper, when we refer to SP, we refer to the name of the actor.

Please cite this article as: Y. Wautelet, M. Kolp, Business and model-driven development of BDI multi-agent systems, Neurocomputing (2016), http://dx.doi.org/10.1016/j.neucom.2015.12.022

Similarly, in order to *Send the Reservation Decision*, the `Client` actor can either achieve the Task *Find SP with a Matchmaker* or *Find Potential SP* (on his own); in either cases he does the tasks *Send Reservation Request* and then *Select Reservation Choice*. Note that the SRD for the *Booking Facilities* service is more elaborated than presented in Fig. 9 but, to keep it readable, we have voluntarily reduced it to the minimum presented, notably in what concerns the traceability from QE and threats that are covered in the rest of this section.

7.2.2. Actors, their accountability and responsibility

At strategic level, we could evaluate actors' accountability; here, at tactical level, we can evaluate actors' responsibility. Indeed, the `Online Platform` actor is, for example, responsible for the achievement of the goal *Get Best Available Offer* meaning that the `Online Platform` has itself to fulfill the process of finding a way of giving the best possible offer to the SP which depends on it. The responsibility evaluation is the same for all the other goals and tasks present in the SRD meaning that the dependee actor has responsibility.

As discussed in the previous section, from the point of view of the Client, the SP is accountable for the *Booking Facilities* Service. The latter Service makes explicit use of the `Online Platform` to achieve the goal *Get Best Available Offer* in the scope of fulfilling this Service. The `Online Platform` could fail to achieve this goal which it is responsible of, this would imply that the quality of service is somehow lowered below an agreed level. Then, the SP is, for the Client, the actor to blame (because as we could see at strategic level it is accountable for it) so that the SP is legally in charge of providing compensation.

This accountability/responsibility assignment can only be determined and viewed thanks to the combination of the strategic and tactical views.

Table 1
How elements are transformed from strategic to operational levels.

Transformation process from strategic to tactical	
Element	Rationale
Actor	At strategic level, the actor is rather a SP or a SC and these are involved in the dependency relationship; they are the main stakeholders for the Service. The SP is accountable for the proper fulfillment of the Service and both the SP and SC will be responsible for performing actions so that the **SP and SC actors will be transformed at tactical level in i* actors**[2]
Service	**Services need for their fulfillment that a few actors pursue goals and softgoals, achieve tasks or furnish resources**. The latter tactical elements can thus all be conceived under the scope of a defined service ensuring traceability
QE and Threats	As seen, QE and threats are by nature non-functional elements. **QE and threats can typically be supported at tactical level by a series of i* softgoals, goals, tasks, and resources that contribute to the satisfaction of a QE or that help lowering the probability of occurrence of the Threat or its impact in case of realization.** Tactical softgoals, goals and tasks can thus be conceived under the scope of a defined QE or Threat ensuring traceability
Transformation process from tactical to operational	
Element	Rationale
Actor	**An actor at tactical level is transformed in an intentional Agent at operational level**. This way an organizational actor is literally mapped to an intelligent acting entity at run-time into the system
Goal & Task	Goals and tasks are functional elements so that these will be transformed into corresponding functions at system level. **Tasks and goals will therefore be transformed into agents' plans**
Resources	A resource is manipulated by an actor but, at the operational system level, it is represented as a conceptual entity; what matters is its possible states so that tactical **resources are transformed into Agent's Beliefs**
Softgoals	Softgoals at tactical level are not transformed at operational level but, as they are satisfied, other (tactical) i* goals and tasks contribute positively or negatively to their satisfaction. These latter goals and tasks are themselves transformed into the agent architecture as evoked above

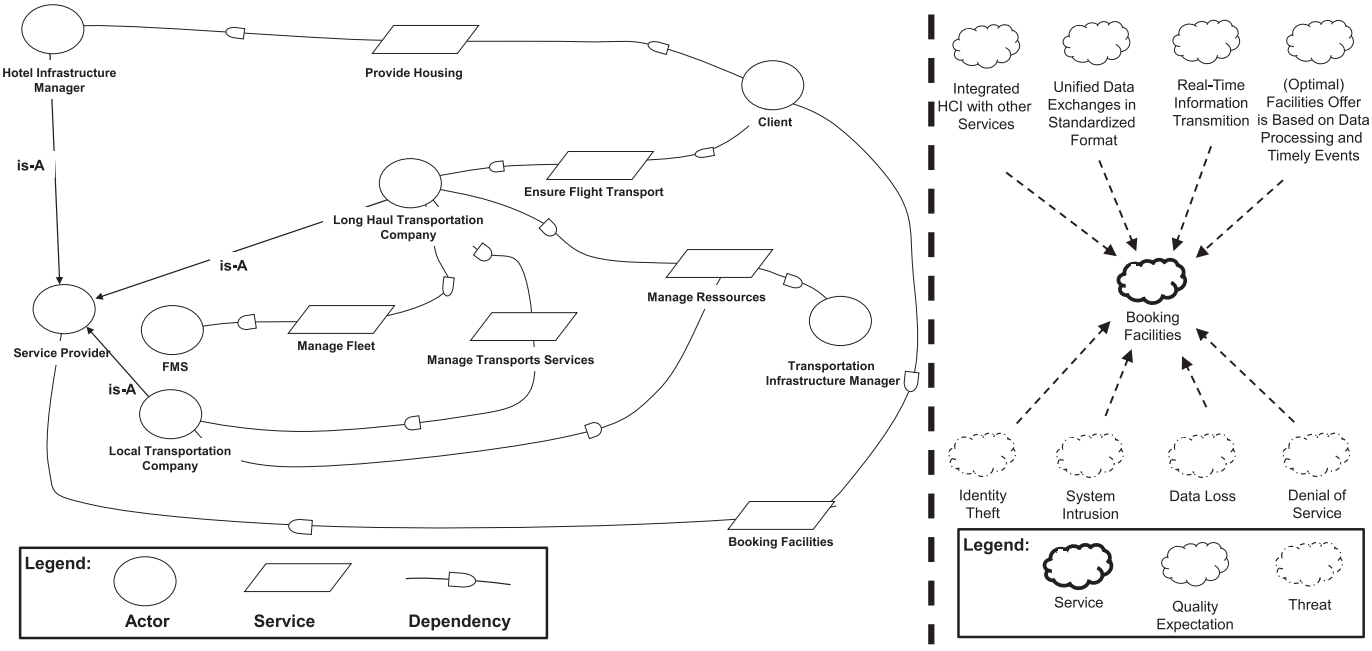

Fig. 8. Collaborative platform for travel planning: strategic layer.

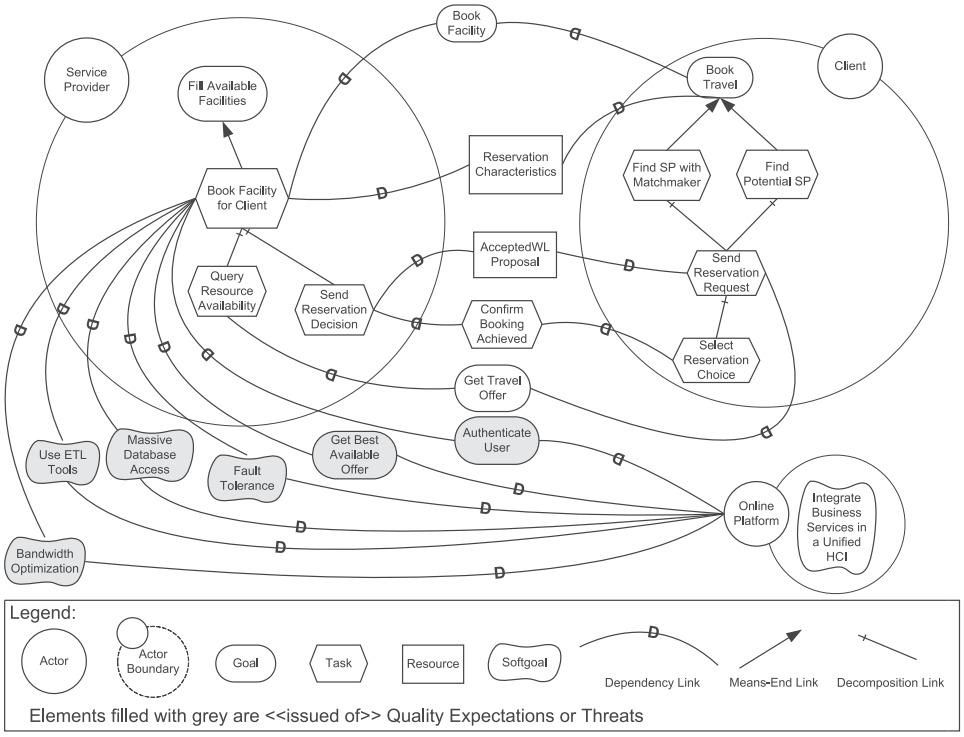

Fig. 9. Strategic rationale diagram: booking facilities service.

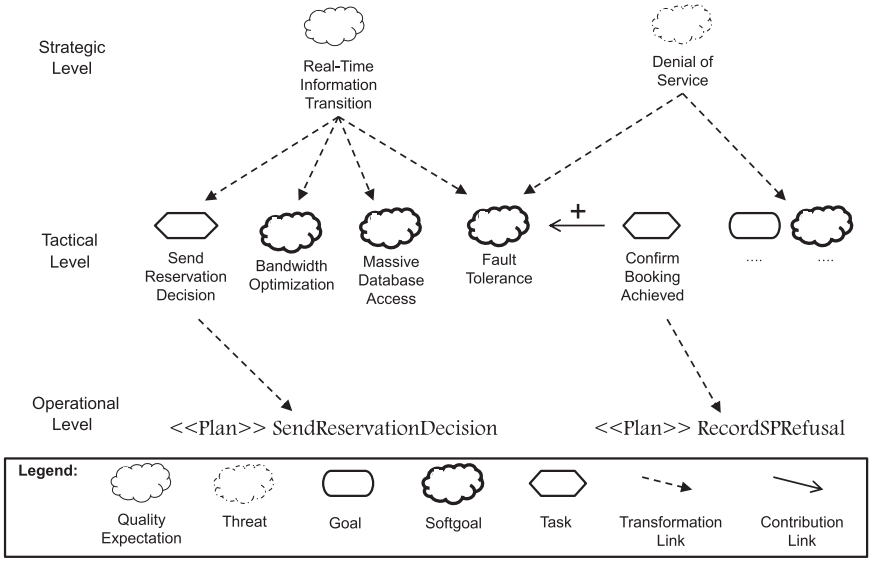

Fig. 10. Transformation process: case study.

7.2.3. Environmental factors

QE and threats are then refined into a set of i* softgoals, goals and tasks at tactical level. Due to a lack of space and to keep the diagrams clear we only illustrate this process for one QE. The QE *Real Time Information Transmission* is supported by the SRD in the form of three i* softgoals, i.e. *Bandwidth Optimization*, *Fault Tolerance* and *Massive Database Access* but also on the i* task *Send Reservation Decision*. The QE *Unified Data Exchanges in Standardized Format* also supported by the SRD through the i* softgoal *Massive Database Access* (so same softgoals can refer to multiple QE) and another i* softgoal: *Use ETL Tools*. As another example, the QE *(Optimal) Facilities Offer is Based on Data Processing and Timely Events* is supported by the softgoals *Massive Database Access* and *Use ETL Tools* but also on the i* goal *Get Best Available Offer*. Let us also take the example of the Threat *System Intrusion* that notably refines in the SRD with the i* goal *Authenticate User*. Similarly, the Threat *Denial of Service* also refines into the i* softgoal *Fault Tolerance*. This process is illustrated in Fig. 10.

The SRD model constitutes the tactical layer of the to-be software application.

7.3. From tactical to operational level: towards agents structure

As said, tactical elements represented into the SRD are forwarded to elements of the agent architecture. The i* actors become agents at operational level and the i* goals and tasks are operationalized through the agent's plans. Finally, the i* resources are operationalized into the agent's beliefs.

Please cite this article as: Y. Wautelet, M. Kolp, Business and model-driven development of BDI multi-agent systems, Neurocomputing (2016), http://dx.doi.org/10.1016/j.neucom.2015.12.022

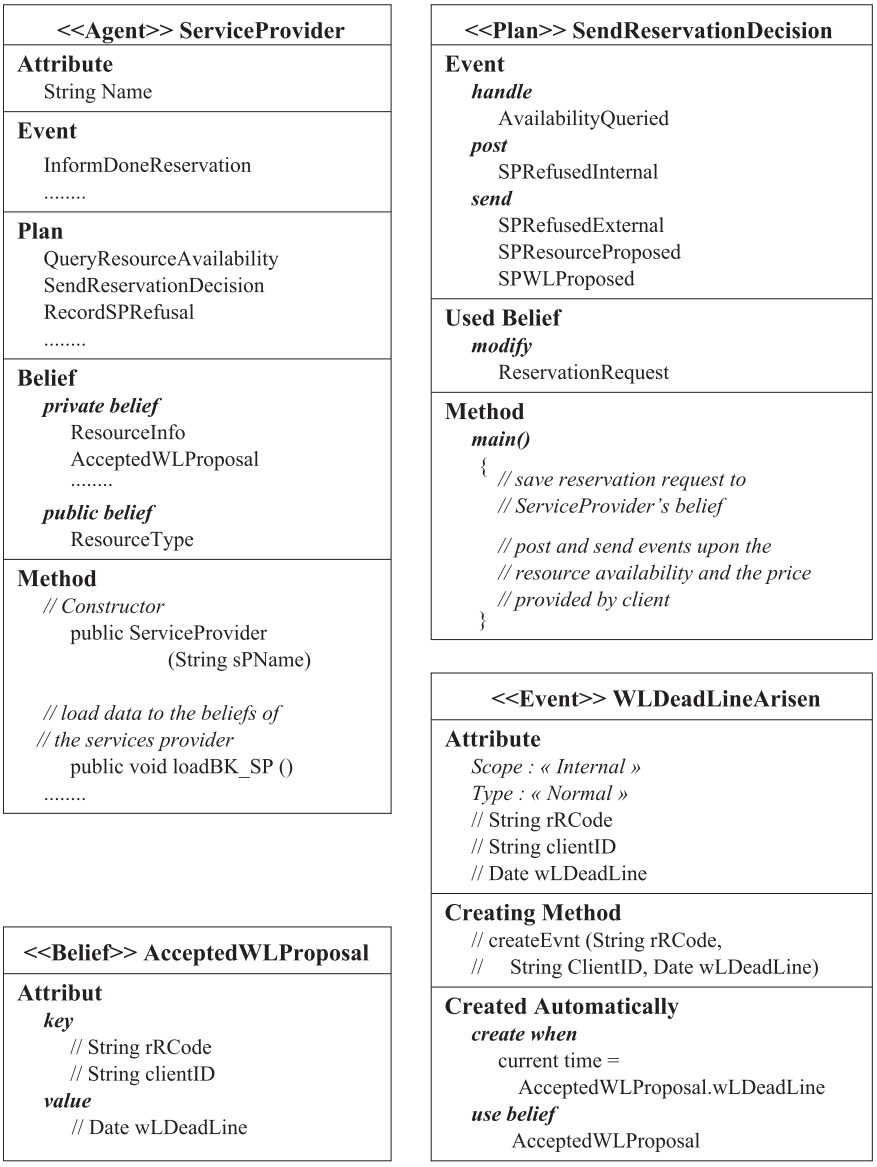

Fig. 11. Structural diagram—booking facilities service.

As an example, Fig. 11 depicts part of the *Booking Facilities* service design. In a few words, each construct described earlier is illustrated only through one component. Each component can be considered as an instantiation of the agent, belief, plan, event templates from Fig. 4.

As shown in Fig. 11, the **ServiceProvider** is one of the two agents composing the *Booking Facilities* service. It uses plans such as `QueryResourceAvailability` which is issued of the Task *Query Resource Availability* of the i* SRD. Another Plan is `SendReservationDecision` issued of the Task *Send Reservation Decision* of the i* SRD. The plan name is thus also the name of the goal or task it operationalizes.

The global belief `ResourceType` is issued of the i* SRD diagram resource called *Reservation Characteristics* used to store the resource type and its descriptions (e.g., for an air ticket booking system, the resource type could be `economic/first class/ business class`; for a hotel room booking system, the resource type could be `single room/double room/ …`). This belief is declared as public since it will be used by both the client agent and the service provider agent. The other beliefs are declared as private since the service provider is the only agent that can manipulate them.

The constructor *method* allows to give a name to a **ServiceProvider** agent when created. This method may call other methods, for example `loadBK_SP()`, to initiate agent beliefs data.

As illustrated in Fig. 10, the **SendReservationDecision** plan is forwarded from the i* SRD Task with the same name and is thus under the scope of the *Booking Facilities* service. It is used by the service provider to answer the Client; more precisely: the `SPRefusedExternal` event is sent when the answer is negative, `SPResourceProposal` when some resource is available for the client, or `SPWLProposal` (Service Provider Waiting List Proposal) when the SP is able to provide the type of resource asked by the client but not at the moment of the request. This plan is executed when the `AvailabilityQueried` event (containing the information about the availability of the resource type required by the client) occurs. This plan also modifies the `ReservationRequest` belief, i.e. the service provider's belief storing the client's reservation request before the service provider sends (or posts) his answer.

The **AcceptedWLProposal** belief is one of the **ServiceProvider**'s beliefs used to store the client's accepted waiting list proposal. The reservation request code `rRCode` and the `clientID` form the belief key. The `reservationInfoCode` attribute that contains the correspondent code of the resource type requested by

Please cite this article as: Y. Wautelet, M. Kolp, Business and model-driven development of BDI multi-agent systems, Neurocomputing (2016), http://dx.doi.org/10.1016/j.neucom.2015.12.022

the client and the `wLDeadLine` that contains the time-out before which the service provider must send a *resource not available* message to the client if no resource is proposed are declared as value fields.

WLDeadlineArisen is an event that is posted automatically whenever the time-out `wLDeadLine` (of the `AcceptedWLProposal` belief) is reached. It will then invoke a plan to inform the client that the resource is not available.

Other agent behavior is outside the scope of the section that focuses on traceability; the agent communication of the case study can be found in Appendix D while the agent dynamic of the case can be found in Appendix E.

8. Conclusion

AOSE methodologies either focus on design models only or lack proper elements for scoping the software project like in Tropos. Indeed, even if the qualities and advantages of high-level abstraction models capturing social and intentional aspects such as the i* strategic dependency and strategic rationale models have been recognized, poor scalability, redundancy, various levels of aggregation and alternative modeling ways make them hard to use in the context of real-life software development. Moreover, for large business software development involving lots of independent actors, responsibly and accountability need to be set-up consistently; a consistent strategic view based on services allows us to deal with these issues.

This work has been an effort to propose a structured model-driven approach to develop agent-oriented software through elements transformation. We thus dispose of an entire structure allowing us to develop models at each structure of decision making each having their specific purpose and abilities. Granularity of elements plays here an important role, we indeed have:

- coarse-grained services constituting the business values of the enterprise as well as the QE and threats they face. Actors' accountability is defined at this level;
- middle-grained goals, tasks and plans constituting the process elements the enterprise requires for service fulfillment. Actors' responsibility is defined at this level;
- fine-grained events and beliefs constituting the structure of agents behavior for operational execution. This constitutes the software run-time.

The horizontal dimension has been illustrated in this paper through the transformation approach. The associated software process also tackles a vertical dimension through an iterative approach. This dimension will be the subject of a future communication.

A case tool called Descartes Architect has been developed to support the approach. It offers multiple views to edit each of the diagrams seen in the transformation process. The tool itself allows us to ensure consistency between the elements traced from one diagram to the other and code generation. Further developments of the tool are under construction.

Appendix A. Event structure

Events describe stimuli, emitted by agents or automatically generated, in response to which other or the same agents must take action.

Events are the origin of all activity within an agent-oriented system. In the absence of events an agent stays idle. Whenever an event occurs, an agent initiates a task to handle it. This task can be thought of as a thread of activity within the agent. The task causes the agent to choose between the plans it has available, executing a plan or a set of plans until it succeeds or fails.

There are different event types, each with different uses. Events can be described along three dimensions:

- *External* or *internal* event: external events are sent to other agents while internal events are posted by an agent to itself.
- *Normal* or *BDI event*: an agent has a number of alternative plans to respond to a BDI event and only one plan in response to a normal event. Whenever an event occurs, the agent initiates a plan to handle it. If the plan execution fails and if the event is a normal event, then the event is said to have failed. If the event is a BDI event, a set of plans can be selected for execution and these are attempted in turn. If all selected plans fail, the event is also said to have failed.
- *Automatic* or *nonautomatic* event: an automatic event is automatically created when certain belief states arise.

The rest of the section studies in more detail these kinds of events and the way agents handle them.

Normal event: Normal events are analogous to events in conventional event-driven programming. That is, they represent transitory occurrences that initiate a single, immediate response from an agent.

When a normal event is received by an agent, the agent initiates a *task* to handle it. This task involves the agent selecting and executing the plan that is *relevant* to this event.

A plan is *relevant* to a given event if it has declaration that matches the event, and its `context`()method succeeds when executed (see the plan structure for more detail). Note that if no plan is found relevant to a particular event, the event is said to have failed. When an event is failed, the system will take care of freeing (or deleting) this event (e.g., the garbage collection technique using by programming languages such as Java).

A `context`() method returns either true or false, meaning that the plan is either applicable or not applicable for a given event.

BDI event: Belief-Desire-Intention (BDI) events represent a different class of event. One important aspect is that it models *goal-directed* behavior in agents. That is, an agent *commits* to the desired outcome, not the method chosen to achieve it.

When receiving a BDI event, an agent selects a number of plans and these are attempted in turn, in order to try to achieve successful plan execution. If the set of chosen plans is exhausted, the event is said to have failed and garbage collectors technique is used to delete this event.

The key difference between normal and BDI events consist in the way an agent selects plans for execution. With normal events, the agent invokes a plan for a given event and executes that plan only. The handling of BDI events is more complex and powerful. An agent can assemble a set of (alternatives) plans for a given event, and try to apply these plans until it achieves its goal. If an agent has more than one relevant plan for a given event, the plans declaration order appearing in the agent declaration will be determining.

Automatic Event: An automatic event allows an agent to automatically post particular events when certain belief states arise. Automatic events use two statements: `create when` and `uses belief`. The `create when` statement specifies the logical condition which must arise

Please cite this article as: Y. Wautelet, M. Kolp, Business and model-driven development of BDI multi-agent systems, Neurocomputing (2016), http://dx.doi.org/10.1016/j.neucom.2015.12.022

for the event to be automatically created. The states of the beliefs that are defined by `uses belief` are monitored to determine when to automatically create events.

The use of these statements to achieve automatic belief monitoring is illustrated in the following example:

```
event BookingLevelTooHigh
...
uses belief BookingTicketsPool wt;
create when(wt.getBookingLevel() > 300)
```

The condition is evaluated once initially, and subsequently whenever a change occurs that might affect the condition.

Event Declarations: As shown in Fig. A1, the structure of an event is composed of three parts:

- the declaration of the attributes of the event;
- the declaration of the methods to create the event;
- the declaration of the beliefs and the condition used for an automatic event.

The event has two particular attributes (in addition to its specific ones): *scope* and *type*. The *scope* attribute captures whether an event is an internal or an external one. The event type (normal or BDI) is captured by the *type* attribute. Each event has a number of methods to provide access to their data. The third part only appears for automatic events.

The declaration of the creation of the event is described after the `Creating Method` statement. `CreatingMethodName(param. list)` is a method that describes how the event can be created. An event's creating method must be used whenever an instance of the event needs to be created. It describes everything that the agent needs to do in order to build an instance of the event.

The `creatingMethod` statement indicates that an event's creating method is being defined. The name by which the creating method is identified is described by the `methodName`. When the agent posts or sends an event, it uses this name to identify the creating method to be used. The parameters identify the number and type of parameters that this creating method requires in order to construct the event. The `creatingMethod` statement that indicates an event's creating service is given below.

A client agent is looking for a plane ticket. The event is posted whenever a ticket is found (we suppose the airline company agent will execute a plan to handle it, e.g., inform the client agent about the availability of the ticket and its price).

```
Event FoundTicketEvent
Attribute
Scope: Internal
Type: Normal
departureDate : Date
returnDate: Date
from: String
to: String
price: Integer
CreatingMethod
foundTicket (dd:Date, rd:Date,
f:String,t:String, p:Integer)
{
departureDate=dd
returnDate=rd
from=f
to=t
price=p
}
```

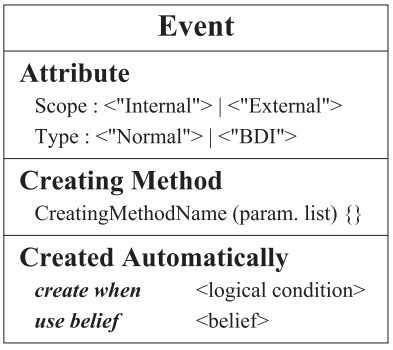

Fig. A1. Event template.

Please cite this article as: Y. Wautelet, M. Kolp, Business and model-driven development of BDI multi-agent systems, Neurocomputing (2016), http://dx.doi.org/10.1016/j.neucom.2015.12.022

In this example, the posting method's name is `foundTicket`. Whenever the agent believes it has found a ticket, it uses the `foundTicket` creating method to create a `FoundTicketEvent`. The `foundTicket` creating method has five arguments: `date`, `return date`, `from`, `to` and `price`. The creating method's body populates the event's data members with this information so that it will be included in the created event.

An event may have one or many creating methods. Different creating methods are used when instances of the event should take a different form under different circumstances.

The declaration of the beliefs and the conditions used for an automatic event are described after the `Created automatically` statement as follows:

`uses belief BeliefType beliefName`

The `uses belief` statement in an event definition has the form shown above. The `BeliefType` refers to the belief of the enclosing agent and `beliefName` is a variable of type `BeliefType`:

`create when < logical condition >`

The `create when` statement has the form shown above. An event definition may include several `created when < logical condition >`.

Note that these two last statements only appear for an automatic event. An example is the `BookingLevelToohigh` event described earlier.

Appendix B. Plan structure

A *plan* describes a sequence of actions that an agent can take when an event occurs.

Plans are structured in three parts as shown in Fig. B1: the *Event* part, the *Belief* part, and the *Method* part. The *Event* part declares events that the plan handles (i.e., events that trigger the execution of the plan) and events that the plan produces. The latter can be either posted (i.e., sent by an agent only to itself) or sent (i.e., sent to other agents). The *Belief* part declares beliefs that the plan reads and those it modifies. The *Method* part describes the plan itself, that is, the actions performed when the plan is executed. In the following, we study in more detail each part of the plan template.

Event `handle < event >`

Whenever an agent detects that an event arises, it tries to find a plan to handle the event.

The event the agent can handle is identified by its plans' `handle < event >` declarations. When an instance of a given event arises, the agent may execute one of the plans that can handle this event.

The `handle < event >` declaration is mandatory. Whenever an instance of this event occurs, the agent will consider this plan as a candidate response. Without a `handle < event >` declaration, a plan would never be executed by an agent.

`post < event >`

The `post < event >` statement declares that the plan is able to post this event when executed. This may be in one of the methods declared in the Method part. Only events that the agent posts internally are declared after the post statement.

`send < event >`

The `send < event >` declaration is similar to the `post < event >` declaration, except that it identifies events that the plan can send to other agents. A plan is able to send an event through one of the methods declared in the Method part.

Used Belief

`read < belief >`

The `read < belief >` declaration indicates that the `< belief >` is to be read (and only read) within the plan.

`modify < belief >`

The `modify < belief >` statement indicates that the `< belief >` may be read and modified within the plan.

Method: All the plan's methods are declared in the method part. Two of these methods are worth to be pointed out: `context()` and `main()`.

An agent may further discriminate plans that handle an event by determining whether a plan is *relevant*. The `context()` method allows the agent to determine which plan to execute when a given event occurs. To be relevant, the plan must declare that it is capable of handling the event that has arisen (via the `handle < event >` declaration) and that it is relevant to the event instance (via the `context` method).

`context()` is a boolean method. If this method returns true, the plan is relevant to the event. If not, the plan is not relevant to the event. This method takes the following form:

```
boolean context (EventType e)
{
//Code to determine if the
//plan is relevant to event e.
}
```

Please cite this article as: Y. Wautelet, M. Kolp, Business and model-driven development of BDI multi-agent systems, Neurocomputing (2016), http://dx.doi.org/10.1016/j.neucom.2015.12.022

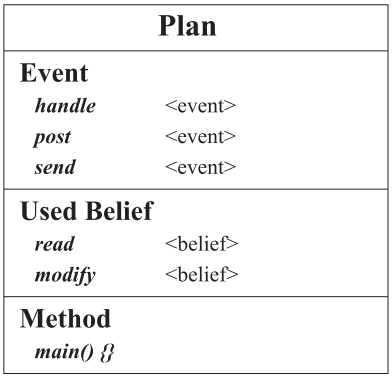

Fig. B1. Plan template.

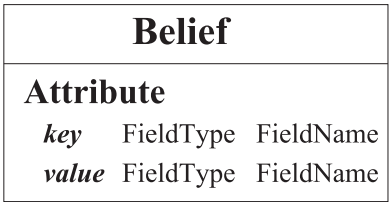

Fig. B2. Belief template.

The `context()` method can also allow the agent to have plans to discriminate between instances of the same event when that event has different values for a given parameter.

For example, suppose that a client agent looking for an air ticket has a plan that describes how to buy a ticket when a ticket is found. Suppose also that the agent includes a `FoundTicketEvent` generated when a ticket is found. The plan would only be relevant when the ticket price is not too expensive (< (2000 for example). The `context()` method can check the value of the `FoundTicketEvent` price attribute and if it is less than $2000, returns `TRUE`.

The `main()` method is executed whenever a plan is executed. The `main()` method is just like the `main()` method in Java – it represents the starting point in a plan's execution.

Belief structure: A *Belief* describes a piece of knowledge that an agent has about itself and its environment.

Fig. B2 shows a template of a belief. Every belief that an agent currently has is represented as tuples. It has key and value fields. The `key FieldType Fieldname` declaration describes the key attributes, the `value FieldType FieldName` declaration the data attributes of each belief. A belief may have zero or more key or value field declarations.

An example of a product belief of a service provider agent may be presented as follows:

```
CloseWorld Product
Attribute
key string productID
value string productName
value double unitPrice
```

An agent's belief can be either `OpenWorld` or `ClosedWorld`. *Closed World* beliefs store true tuples and assume any tuple not found is false. *Open World* beliefs store true and false tuples and assume any tuple not found is unknown.

Appendix C. Concurrent communication

To support concurrent events, Fig. C1 depicts three ways of expressing multiple events. Fig. C1(a) indicates that all events `Event-1` to `Event-n` are concurrently sent. Fig. C1(b) includes a decision box indicating that a decision box will decide which Events (zero or more) will be sent. If more than one Event is sent, the communication is concurrent. In short, it indicates an `inclusive or`. Fig. C1 (c) indicates an `exclusive or`, so that exactly one Event will be sent. Fig. C1(a) indicates an `and` communication.

Fig. C2 illustrates how to use the concurrent events depicted in Fig. C1. Fig. C2(a) and C2(a') portrays two ways of expressing concurrent events sent from an agent to another one. The multiple vertical, or *activation*, bars indicate that the receiving agent is processing the various events concurrently. Fig. C2(a) displays parallel activation bars and Fig. C2(a') activation bars that appear on top of each other.

A few things should be noted about these two variations:

- The semantic meaning is equivalent; the choice is based on ease and clarity of visual appearance (e.g., Fig. C2(a) is bigger but clearer than Fig. C2(a') in which the time order is somewhat ambiguity).
- These figures indicate that a single agent is concurrently receiving the multiple Events. However, the concurrent Events could *each* have been sent to a *different* agent, e.g., `Event-1` to `agent-2`, `Event-2` to `agent-3`, and so on.

Please cite this article as: Y. Wautelet, M. Kolp, Business and model-driven development of BDI multi-agent systems, Neurocomputing (2016), http://dx.doi.org/10.1016/j.neucom.2015.12.022

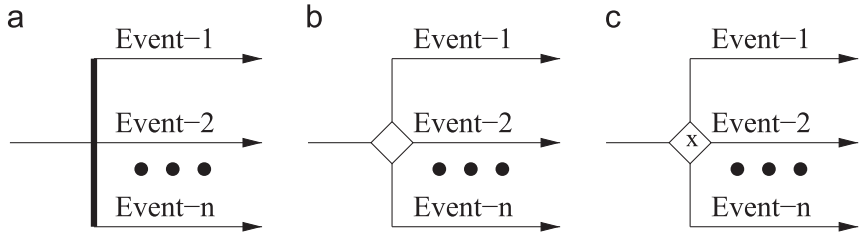

Fig. C1. Expressing multiple events.

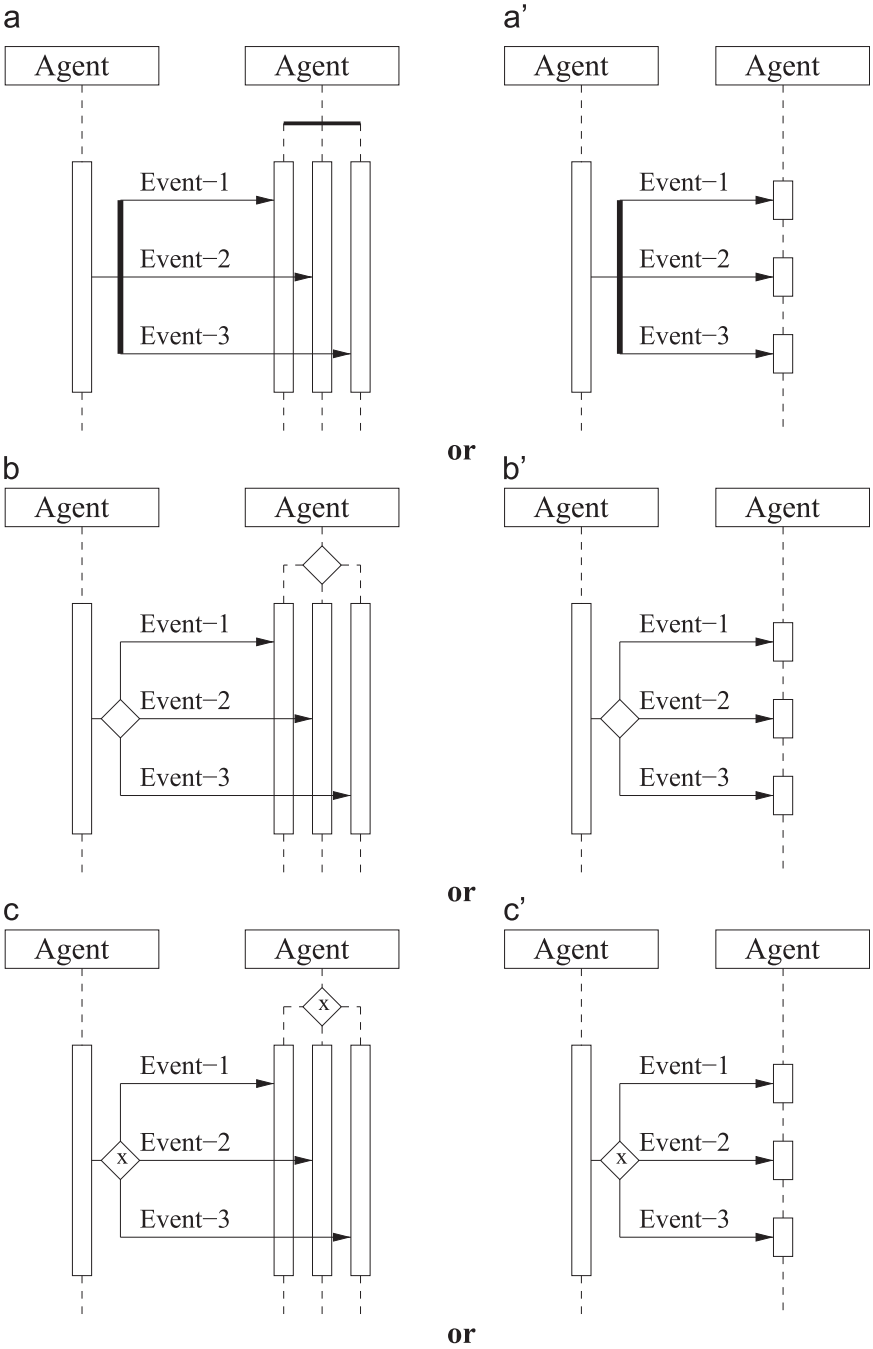

Fig. C2. Basic format for the communication dimension.

Please cite this article as: Y. Wautelet, M. Kolp, Business and model-driven development of BDI multi-agent systems, Neurocomputing (2016), http://dx.doi.org/10.1016/j.neucom.2015.12.022

Appendix D. Agent communication in the travel planning case study

Fig. D1 shows the sequence diagram for our Booking service. The client (`customer1`) sends a reservation request (`ReservationRequestSent`) containing the characteristics (place, room, etc.) of the resource it wishes to obtain from service providers. The service provider may alternatively answer with a denial (`SPRefusedExternal`), a waiting list proposal (SPWLProposed) or an approval, i.e, a resource proposal when there exists such a resource that satisfies the characteristics the client sent.

In case of a waiting list proposal (`SPWLProposed`), when the client accepts it (`AcceptedWLProposalSent`), it sends a waiting list time-out (`wLDeadLine`) to the service provider. Before reaching the time-out, the service provider must send a refusal to the client, in the case it does not find an available slot in the waiting list (`ResourceNotAvailableMessageSent`), or propose a resource to the client. In the latter case, the interaction continues as in the case that the resource proposal is sent to the client.

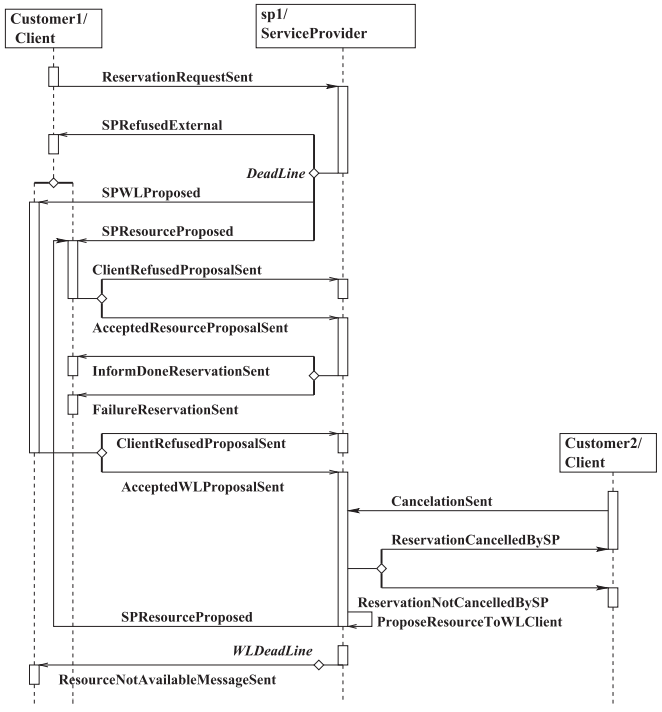

Fig. D1. Communication diagram—booking facilities service.

A resource that is not available becomes available when some client (`customer2` in Fig. D1) cancels its reservation.

Appendix E. Agent dynamic in the travel planning case study

Fig. E1 depicts a dynamic diagram of the *Booking Facilities* service. The diagram models the flow of control from the emission of a reservation request to the reception by the client of the answer from the service provider (*Refusal*, *Resource Proposal*, or *Waiting List Proposal*). Two swimlanes, one for each agent of the *Booking Facilities* service, compose the diagram.

In this example, the `FindPotentialSP` plan is operationalized either by the `FindSP` or the `FindSPWithMM` plans (the client finds potential service providers based on its own knowledge or via a matchmaker).

`MaxPrice` stores the maximum value that the client can afford to obtain a resource unit; `quantity` stores the number of resource units the client wishes to book; `reservedQuantity` and `maxQuantity` respectively store the actual quantity of resource units that are reserved and the maximum number of resource units that the service provider can provide.

Please cite this article as: Y. Wautelet, M. Kolp, Business and model-driven development of BDI multi-agent systems, Neurocomputing (2016), http://dx.doi.org/10.1016/j.neucom.2015.12.022

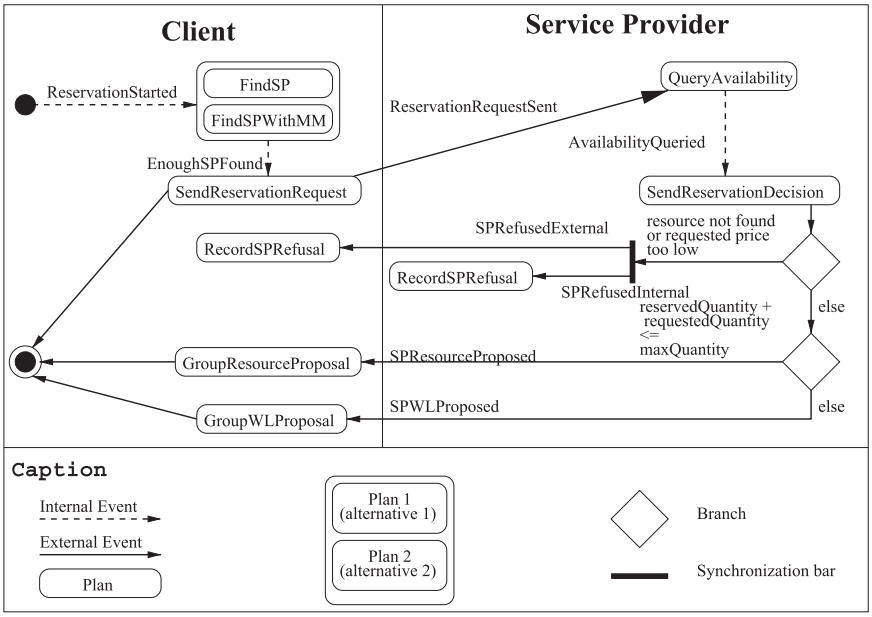

Fig. E1. Dynamic diagram—booking facilities service.

References

[1] IBM, The Rational Unified Process, Version 7.0.1, 2007.
[2] I. Jacobson, G. Booch, J. Rumbaugh, The Unified Software Development Process (1999).
[3] I. Jacobson, S. Bylund, The Road to the Unified Software Development Process (2000).
[4] P. Kruchten, The Rational Unified Process : An Introduction (2003).
[5] L. Motiwalla, J. Thompson, Enterprise Systems for Management, Prentice-Hall, Upper Saddle River, New Jersey, 2012.
[6] Office of Government Commerce, Managing Successful Projects with PRINCE2, Stationery Office Books, London, 2009.
[7] J. Castro, M. Kolp, J. Mylopoulos, Towards requirements-driven information systems engineering: the tropos project, Inf. Syst. 27 (6) (2002) 365–389.
[8] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, J. Mylopoulos, Tropos: an agent-oriented software development methodology, Auton. Agents Multi-Agent Syst. 8 (3) (2004) 203–236.
[9] L. Penserini, A. Perini, A. Susi, J. Mylopoulos, High variability design for software agents: Extending tropos, TAAS 2 (4) (2007) 27.
[10] M. Kolp, P. Giorgini, J. Mylopoulos, Multi-agent architectures as organizational structures, Autonom. Agents Multi-Agent Syst. 13 (1) (2006) 3–25.
[11] J. Mylopoulos, J. Castro, M. Kolp, The evolution of tropos, in: J. Bubenko Jr., J. Krogstie, O. Pastor, B. Pernici, C. Rolland, A. Sølvberg (Eds.), Seminal Contributions to Information Systems Engineering, Springer, Berlin Heidelberg, 2013, pp. 281–287.
[12] O. Pastor, H. Estrada, A. Martínez, The Strengths and Weaknesses of the i* Framework: An Experimental Evaluation, MIT Press, Cambridge, USA (2011), p. 607–645 , Chapter 18.
[13] E. Yu, P. Giorgini, N. Maiden, J. Mylopoulos, Social Modeling for Requirements Engineering, MIT Press, Cambridge, Massachusetts, 2011.
[14] E. Yu, Modeling Strategic Relationships for Process Reengineering, MIT Press, Cambridge, USA, 2011, pp. 1–153 (Chapters 1–2).
[15] E. S. K. Yu, Social modeling and i*, in: A. Borgida, V.K. Chaudhri, P. Giorgini, E.S. K. Yu (Eds.), Conceptual Modeling: Foundations and Applications - Essays in Honor of John Mylopoulos, Lecture Notes in Computer Science, vol. 5600, Springer, Berlin Heidelberg, 2009, pp. 99–121.
[16] E.S.K. Yu, Towards modeling and reasoning support for early-phase requirements engineering, in: 3rd IEEE International Symposium on Requirements Engineering (RE'97), IEEE Computer Society, Annapolis, MD, USA, January 5–8, 1997, pp. 226–235.
[17] M.E. Bratman, Intention, Plans, and Practical Reason, Cambridge University Press, Cambridge, 1999.
[18] A.S. Rao, M.P. Georgeff, BDI agents: From theory to practice, in: V.R. Lesser, L. Gasser (Eds.), Proceedings of the First International Conference on Multiagent Systems, San Francisco, CA, USA, The MIT Press, June 12–14, 1995, pp. 312–319.
[19] A. Casali, L. Godo, C. Sierra, A graded BDI agent model to represent and reason about preferences, Artif. Intell. 175 (7-8) (2011) 1468–1478.
[20] A. Pokahr, L. Braubach, C. Haubeck, J. Ladiges, Programming BDI agents with pure java, in: J.P. Müller, M. Weyrich, A.L.C. Bazzan (Eds.), Multiagent System Technologies—12th German Conference, MATES 2014, Stuttgart, Germany, Proceedings, Lecture Notes in Computer Science, vol. 8732, Springer, September, Cham, 23–25, 2014, pp. 216–233.
[21] S. Kiv, Y. Wautelet, M. Kolp, Agent-driven integration architecture for component-based software development, Trans. Comput. Collect. Intell. 8 (2012) 121–147.
[22] M. Wooldridge, An Introduction to MultiAgent Systems, 2nd ed., Wiley Publishing, Glasgow, 2009.
[23] Y. Wautelet, Representing, modeling and engineering a collaborative supply chain management platform, IJISSCM 5 (3) (2012) 1–23.
[24] Y. Wautelet, M. Kolp, S. Poelmans, Requirements-driven iterative project planning, in: Software and Data Technologies, 2013, pp. 121–135.
[25] F. Bellifemine, G. Caire, D. Greenwood, Developing Multi-agent Systems with JADE, Vol. 5, Wiley, Chichester, 2007.
[26] S. Poslad, Specifying protocols for multi-agent systems interaction, TAAS 2 (4). 2007, http://dx.doi.org/10.1145/1293731.1293735.
[27] J.M. Gascuen`a, E. Navarro, A. Fernández-Caballero, Model-driven engineering techniques for the development of multi-agent systems, Eng. Appl. AI 25 (1) (2012) 159–173.
[28] J. Pavón, J. J. Gómez-Sanz, R. Fuentes, Model driven development of multi-agent systems, in: A. Rensink, J. Warmer (Eds.), ECMDA-FA, Lecture Notes in Computer Science, vol. 4066, Springer, Berlin Heidelberg, 2006, pp. 284–298.
[29] M. Winikoff, L. Padgham, Developing Intelligent Agent Systems: A Practical Guide, Halsted Press, New York, NY, USA, 2004.
[30] C. Hahn, C. Madrigal-Mora, K. Fischer, A platform-independent metamodel for multiagent systems, Autonom. Agents Multi-Agent Syst. 18 (2) (2009) 239–266.
[31] I. Ayala, M. Amor, L. Fuentes, model driven engineering process of platform neutral agents for ambient intelligence devices, Autonom. Agents Multi-Agent Syst. 28 (2) (2014) 214–255, http://dx.doi.org/10.1007/s10458-013-9223-3.
[32] N. Howden, R. Rönnquist, A. Hodgson, A. Lucas, Intelligent agents—summary of an agent infrastructure, in: 5th International Conference on Autonomous Agents, 2001.
[33] T. Abdelaziz, M. Elammari, R. Unland, C. Branki, Masd: multi-agent systems development methodology, Multiagent Grid Syst. 6 (1) (2010) 71–101.
[34] The Descartes Architect Case-tool, 2015, ⟨http://www.isys.ucl.ac.be/descartes/⟩.
[35] P. Kruchten, The 4+1 view model of architecture, IEEE Softw. 12 (6) (1995) 42–50.
[36] A. Susi, A. Perini, J. Mylopoulos, P. Giorgini, The tropos metamodel and its use, Informatica (Slovenia) 29 (4) (2005) 401–408.
[37] H. Estrada, A.M. Rebollar, O. Pastor, J. Mylopoulos, An empirical evaluation of the ∗ framework in a model-based software generation environment, in: E. Dubois, K. Pohl (Eds.), Advanced Information Systems Engineering, 18th International Conference, CAiSE 2006, Proceedings, Lecture Notes in Computer Science, vol. 4001, Springer, Luxembourg, June 5–9, 2006, pp. 513–527.
[38] R. Haesen, M. Snoeck, W. Lemahieu, S. Poelmans, On the definition of service granularity and its architectural impact, in: Z. Bellahsene, M. Léonard (Eds.),

Please cite this article as: Y. Wautelet, M. Kolp, Business and model-driven development of BDI multi-agent systems, Neurocomputing (2016), http://dx.doi.org/10.1016/j.neucom.2015.12.022

CAiSE, Lecture Notes in Computer Science, vol. 5074, Springer, Berlin Heidelberg, 2008, pp. 375–389.
[39] R. Ferrario, N. Guarino, Towards an ontological foundation for services science, in: J. Domingue, D. Fensel, P. Traverso (Eds.), Lecture Notes in Computer Science, FIS, vol. 5468, Springer, Berlin Heidelberg, 2008, pp. 152–169.
[40] L. Liu, E. Yu, From requirements to architectural design using goals and scenarios, in: Proceedings of the 1st International Workshop from Software Requirements to Architectures. Toronto, Canada, 2001, pp. 22–30.
[41] L. Chung, B. Nixon, E. Yu, J. Mylopoulos, Non-Functional Requirements in Software Engineering, Kluwer Academic Publishing, New York, 2000.
[42] A. van Lamsweerde, Requirements Engineering: From System Goals to UML Models to Software Specifications, Wiley, Glasgow, 2002.
[43] A. Pokahr, L. Braubach, W. Lamersdorf, Jadex: A BDI reasoning engine, in: R.H. Bordini, M. Dastani, J. Dix, A.E. Fallah-Seghrouchni (Eds.), Multi-Agent Programming: Languages, Platforms and Applications, Multiagent Systems, Artificial Societies, and Simulated Organizations, vol. 15, Springer, New York, 2005, pp. 149–174.
[44] A. Pokahr, L. Braubach, K. Jander, The jadex project: Programming model, in: M. Ganzha, L.C. Jain (Eds.), Multiagent Systems and Applications—Vol. 1: Practice and Experience, Intelligent Systems Reference Library, vol. 45, Springer, Berlin Heidelberg, 2013, pp. 21–53.
[45] B. Bauer, J.P. Müller, J. Odell, Agent uml: A formalism for specifying multiagent software systems, Int. J. Softw. Eng. Knowl. Eng. 11 (3) (2001) 207–230.
[46] L. Cabac, D. Moldt, Formal semantics for auml agent interaction protocol diagrams, in: J. Odell, P. Giorgini, J.P. Müller (Eds.), AOSE, Lecture Notes in Computer Science, vol. 3382, Springer, Berlin Heidelberg, 2004, pp. 47–61.

Yves Wautelet is an Assistant Professor in Information Systems at KU Leuven, Belgium. He formerly has been an IT project manager and a Postdoc Fellow at the Universitécatholique de Louvain, Belgium. He completed a Ph.D. thesis focusing on project and risk management issues in large enterprise software design. Yves also holds a Master of Management Sciences as well as a Master of Information Systems. His research interests include aspects of software engineering and enterprise information systems such as requirements engineering, agent-oriented programming and COTS-based development. He also focuses on the application of his research into industrial environments.

Manuel Kolp is a Full Professor in IT (Information Systems) and Vice-Dean at the Louvain School of Management (LSM), Universitécatholique de Louvain (UCL), Belgium. He was a Post Doctoral Research Associate in Computer Science at the University of Toronto, Canada, for a couple of years and worked there with the Department of Computer Science in Requirements Management and Multi-Agent Software Engineering. Previously, he was a FNRS (Belgian National Agency for Scientific Research) research fellow in Knowledge Representation and Object Oriented Information Systems at UCL and received his Ph.D. degree from the University of Brussels. Manuel Kolp has also been collaborating as a lead investigator on projects dealing with knowledge, information and data systems or e-business and ERP II applications and regularly acts as an IT expert and advisor for enterprises and organizations.

Please cite this article as: Y. Wautelet, M. Kolp, Business and model-driven development of BDI multi-agent systems, Neurocomputing (2016), http://dx.doi.org/10.1016/j.neucom.2015.12.022