On ISOA: Intentional Services Oriented Architecture

Colette Rolland[1], Rim Samia Kaabi[1], and Naoufel Kraiem[2]

[1] Université Paris1 Panthéon Sorbonne, 90, rue de Tolbiac, 75013 Paris, France
[2] Ecole Nationale des Sciences de l'Informatique, 2035 Manouba, Tunis, Tunisia
rolland@univ-paris1.fr, rim-samia.kaabi@malix.univ-paris1.fr, Naoufel.Kraiem@ensi.rnu.tn

Abstract. Despite the growing acceptance of SOA, service-oriented computing remains a computing mechanism to speed-up the design of software applications by assembling ready-made services. We argue that it is difficult for business people to fully benefit of the SOA if it remains at the software level. The paper proposes a move towards a description of services in business terms, i.e. intentions and strategies to achieve them and to organize their publication, search and composition on the basis of these descriptions. In this way, it leverages on the SOA to an intentional level, the ISOA. We present $\mathcal{ISM}$, the model to describe intentional services, and to populate the service registry. We highlight its intention driven perspective for service description, retrieval and composition. Thereafter, we propose a methodology to determine intentional services that meet business goals. Finally, we introduce agent architecture to support model driven execution of intentional services.

Keywords: Service-oriented computing, service-oriented architecture, intentional service-oriented architecture, intentional service modelling, intention-driven service composition.

1 Introduction

Service-Oriented Computing (SOC) is the computing paradigm that utilizes services as fundamental elements for developing software applications [1][2]. SOC relies on the Service-Oriented Architecture (SOA) [3] that is a way of reorganizing a portfolio of previously developed applications into services that are self-describing, platform agnostic computational elements performing functions, accessible through standard interfaces and that can be assembled in complex compositions based on standard messaging protocols. As shown in Fig.1, the basic SOA defines an interaction between three kinds of software agents [4], namely, the *service provider*, the *service client* and the *service registry* involving the *publish, find* and *bind* operations. Services are offered by *service providers* that procure the service implementations and supply their descriptions to a service registry. The service registry *publishes* services by making their descriptions. The service client uses the *find* operation to retrieve the service description matching his functional needs and uses it to *bind* with the service provider and invoke the service.

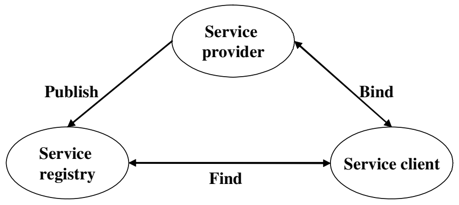

Fig. 1. Function-driven SOA

SOA is a way of designing a software system that is *function-driven*. Services perform functions implemented in software, wrapped with formal documented interfaces which provide the mechanism by which services can communicate with one another in compositions to perform higher level functions. The service interface (that provides the signatures of the available operations) is central to the SOA view as it is the only thing, which is exposed to the client to invoke the function.

However, it shall be noticed that interface descriptions are low level, technical statements (cf. WSDL statements [5]) that are understandable by software professionals but far to be comprehensible by business people. At the same time, the notion of a service is familiar to the management world [6] and with the growing acceptance and popularity of SOA, computing systems now aim to extend far beyond the firewall to automate enterprise-wide business processes, covering sales, supply chain, manufacturing, delivery, payment, human resources, and more. To attain this, it is necessary to adapt SOA to a mainstream practitioners' level and bridge the gap between high level business services and low level software services [7], [8].

The position adopted in this paper is to suggest a move from the *function-driven* SOA to *intention-driven* SOA. Whereas the former lies on a functional view of services, the latter proposes to spell out the purpose, the intention behind a service. As a consequence, interfaces of these services will bring out the business goal that the service allows to fulfil instead of defining the signatures of basic operations that can be invoked on class objects. This will avoid the current mismatch of languages between low level services expressions such as WSDL statements and business perceived services. We refer to these services as *intentional services* and present in this paper *ISM*, a model for intentional service modelling.

While complying with the SOA model, our model, the intentional SOA, ISOA is a proposal for leveraging the 3-SOA tuple <Publish, Find, Bind> to an intentional level matching the business mainstream needs. In adapting the roles and operations of the SOA model, the ISOA (Fig. 2) introduces two main departures:

(i) in the interaction, business agents replace software agents,
(ii) intentional service descriptions replace functional software service descriptions.

The ISOA implies that business centric organizations offering e-business services shall describe their services in an intentional manner, and publish them to an e-business service registry that makes these descriptions available in an intentional service registry. Business agents who are searching for services use an intention matching mechanism to retrieve service descriptions fitting their needs and use them to bind to the e-business provider.

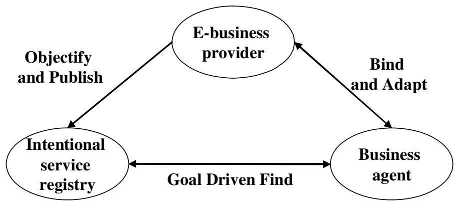

Fig. 2. Intention-driven SOA

In this paper, we use the three roles mentioned in the ISOA architecture to structure the discussion on ISOA. For the registry, we introduce the notion of *intentional service*, highlight its relationship with software services and present *ISM*, a model for intentional service modelling. We show that an intentional service description shall include variability, i.e. propose alternative variations of a given component service. The model gives to us the capability to populate the intentional service registry. This is the subject of section 2.

It is for the e-business provider to define the services that are to be provided in the business. We propose to represent business intentions in a graphical representation called *Map*. This Map takes the form of an intention/strategy graph with intentions as nodes and strategies to achieve them as edges. The e-business provider derives the services that can be published from a map following a set of guidelines. This role of the provider is considered in section 3.

Finally, in carrying out his role, the business agent must be provided with the appropriate execution architecture particularly to handle variability. In section 4, we outline an agent architecture for service execution.

2 Populating the Registry with Intentional Services

In this section we consider the Intentional Service Registry. The aim is to develop a model that defines the contents of the Registry. Towards this end, we clarify the notion of an intentional service and present the Intentional Service Model, *ISM*, to model different types of intentional services.

2.1 Intentional Service Model

An *intentional service* is a service captured at the business level, in business comprehensible terms and described in an intentional perspective, i.e. focusing on the intention it allows to achieve rather than on the functionality it performs. Fig. 3 presents *ISM* using UML notations. As shown by the colors used in the Figure, there are three different aspects in the description of an intentional service, namely the service interface, the service behavior and the service composition. We describe the three in turn.

First, central to the Figure is the fact that a service permits the fulfillment of an *intention,* given an *initial situation* and terminating in a *final situation.* These three elements constitute the interface of an intentional service; the intention replaces the

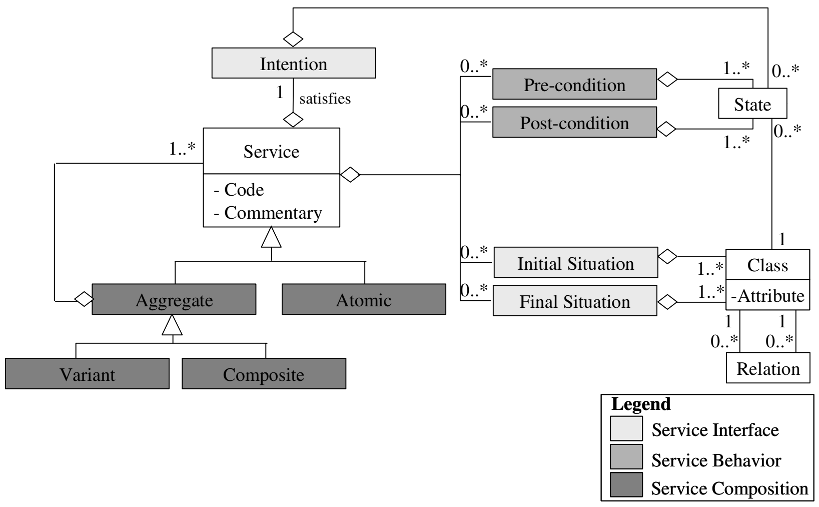

Fig. 3. Intentional Service Model

operations that are part of a typical software interface whereas the initial and final situations are the input and output parameters structured as business object classes.

We view an intention in the same sense as a goal. A goal is 'an optative' statement [9], that expresses what is wanted i.e. a state that is expected to be reached or maintained. Thus, *Make Room Booking* is the intention to make a reservation for rooms in a hotel. The achievement of this intention leaves the system in the state, *Booking made.* If *Accept Payment* is the intention of a service then the initial situation refers to the *booking* and *customer* classes whereas the final situation comprises the *payment* class in addition.

Second, the behavior of the service is specified through its *pre* and *post conditions* that are the initial and final sets of states characterizing the initial and the final situation respectively. In the *Accept Payment* service example, <booking.state ='OK' ∧ customer.status='registered'> and <booking.state='paid' ∧ payment.status = 'done' > are the pre and post-conditions respectively.

Finally, services are classified as *aggregate* or *atomic*. The former are composed of other services whereas the latter are not. *Atomic services* have intentions that are fulfilled by SOA level functional services. In contrast *aggregate services* have high-level intentions that need to be decomposed in lower level ISOA services till atomic intentional services are found. Therefore, it can be understood that aggregate intentional services lie on an intention-driven composition that is necessary to bridge the gap between the actual functionality (captured in the atomic service) and the high level perception of business executives for a service fulfilling their strategic/tactical intentions.

Fig. 3 shows that aggregate services are further refined. Aggregation of services can involve *variants*, i.e. services which are alternative to the others or result from simple composition, leading to *composite services*.

Composite services reflect the precedence/succession relationship between their intentions. For example, in the room booking case, *Make Room Booking* must precede

Accept Payment. The composition of this two services leads to the satisfaction of the intention *Make Confirmed Booking*. This form of composition is grounded on the AND goal decomposition as used in goal modelling [10].

The composition is denoted "•" when there is a sequential order between component services and "//" when they can run in parallel. Every service in a composition can be executed repeatedly, this is denoted by the "*" symbol. Thus, the composite service to fulfil the *Make Confirmed Booking* intention is defined as follows:

$$S_{\textit{Make Confirmed Booking}} = \bullet(S_{\textit{Make Room Booking}}, S_{\textit{Accept Payment}})$$

Introduction of *variability* in intentional service modelling is justified by the need to introduce flexibility in intention achievement and adaptability in intentional service execution. There are three types of variants in *ISM*, namely *alternative, choice* and *multi-path.*

An *alternative* variation corresponds to an XOR relationship between the service intentions involved. For example, assume that *Accept Payment* can be achieved in exclusively one of the following ways, *By electronic transfer* or *By credit card* or *By cash.* This leads to define the service $S_{\textit{Accept Payment}}$ as a variant aggregate with three alternative components. We use the symbol "⊗" to denote alternative and therefore:

$$S_{\textit{Accept Payment}} = \otimes(S_{\textit{Accept Payment by electronic transfer}}, S_{\textit{Accept Payment by credit card}}, S_{\textit{Accept Payment by cash}})$$

A *choice* variation corresponds to an OR relationship between the service intentions involved. For example, assume that *Investigate Candidate Booking* can be achieved either *On the Internet* or *By visiting a travel agent* or by both. The aggregate service $S_{\textit{Investigate Candidate Booking}}$ is therefore defined as variant with two components $S_{\textit{Investigate Candidate Booking on the Internet}}$ and $S_{\textit{Investigate Candidate Booking by visiting a travel agent}}$. We use the symbol "**v**" to denote the choice variation and therefore:

$$S_{\textit{Investigate Candidate Booking}} = v\,(S_{\textit{Investigate Candidate Booking on the Internet}}, S_{\textit{Investigate Candidate Booking by visiting a travel agent}})$$

Finally a *multi-path* variation occurs when several compositions of an intentional service allow to achieve the same intentional service. Let us assume in our example that it is possible that the customer gets a booking as a reward for loyalty to the hotel chain. Thus, there are two paths to providing the intentional service *Make a Confirmed Booking:* one by achieving the sequence of intentional services *Make a Booking***,** *Accept payment* and the other one *Get a Rewarded Booking*. The multi-path is denoted "∪" and the multi-path service $S_{\textit{Make Confirmed Booking}}$ is defined as follows:

$$S_{\textit{Make Confirmed Booking}} = \cup\,(\bullet(S_{\textit{Make Room Booking}}, S_{\textit{Accept Payment}}), S_{\textit{Get a Rewarded Booking}})$$

The foregoing demonstrates that services are defined recursively; an aggregate service being possibly composed of other aggregate services; besides, components of an aggregate service can be related directly through composition links (•, //, *) or in a more complex manner through relationships (∪,⊗,v). Relationships between intentional services introduce variability in the composition. Overall, services are defined in an intention-driven manner focusing on the '*whys*' of the functionality provided by the underlying SOA level software service. Moreover, composition is itself intention-driven and grounded in XOR, OR, AND relationships among intentional services. Thus, whereas the service interface exhibits the '*whys*' of the service, its actual implemented functionality is embedded in the related atomic services.

Now, consider the issue of populating the Intentional Service Registry. Evidently, every service must be available in the Registry. That is, every atomic and aggregate service is kept here. For an aggregate, information about composition links and relationships is kept. This enables (Fig. 2) retrieval of complete aggregate services, their binding and adaptation to conform to the task at hand. Retrieval is based on intention matching and thereafter on situation and condition matching. That is, given the need to find a service with intention *I*, the registry is searched to retrieve a service with the same or similar intention. Once such a service is found, one drills down to assure oneself that the pre and post conditions match. Finally, the initial and final situations yield the input and output parameters.

3 Discovering Services for Publication

We believe that the services that populate the Registry arise in the business of organizations. Services to be provided relate to business objectives and, indeed, help to achieve these. This requires that a model of the business can be developed using which the E-business provider (Fig. 2) discovers services for publication. In this section, we propose the use of the Map formalism [11] to represent businesses in intentional terms and provide guidelines to determine services from this representation. We use Materials Management (MM) to illustrate service publication (see [11] for full details of the MM map).

3.1 Capturing Business Intentionality in Maps

Map is a representation system that was originally developed to represent a process model expressed in intentional terms. It provides a representation mechanism based on a non-deterministic ordering of *intentions* and *strategies*. We will use it here as a means for modelling intention-driven composition of services.

A *map* is a labelled directed graph with *intentions* as nodes and *strategies* as edges. An edge enters a node if its strategy can be used to achieve the intention of the node. There can be multiple edges entering a node.

An *intention* is a goal that can be achieved by the performance of a process. For example, the MM map in Fig. 4 has *Purchase Material* and *Monitor Stock* as intentions. Furthermore, each map has two special intentions, *Start* and *Stop*, to respectively start and end the process.

A *strategy* is an approach, a manner to achieve an intention. In Fig.4, *By reorder point planning* is a manner to place an order to *Purchase Material,* any time the stock of this material falls under the reorder point.

A *section* is the key element of a map. It is a triplet as for instance <*Start, Purchase Material, Manual Strategy*> which couples a source intention (*Start*) to a target intention (*Purchase Material*) through a strategy (*Manual strategy*) and represents a way to achieve the target intention *Purchase Material* from the source intention *Start* following the *Manual Strategy*.

Sections in a map are related to each other by four kinds of relationships namely *multi-thread*, *bundle*, *path* and *multi-path* relationships.

Bundle relationship: Several sections having the same pair of source and target intention, which are mutually exclusive are in a *bundle relationship*. For example in Fig.4, the *Planning strategy* is a bundle consisting of the *Reorder point strategy* and *Forecast based strategy*. Similarly, the *Inventory balance strategy* is a bundle of *periodic*, *continuous* and *sampling* strategies.

Multi-thread relationship: It is possible for a target intention to be achieved from a source intention in many different ways. Each of these ways is expressed as a section in the map and these sections are in a *multi-thread relationship* with one another. In Fig.4 the *Planning strategy* and the *Manual strategy* are in a *multi-thread* relationship. The difference between a *multi-thread* and a bundle relationship is that of an exclusive OR of sections in the latter versus an OR in the former.

Path relationship: This establishes a precedence/succession relationship between sections. For a section to succeed another, its source intention must be the target intention of the preceding one. For example the two sections, *<Start, Purchase Material, Manual strategy >, <Purchase Material, Monitor Stock, Out-In strategy >* constitutes a *path*.

Multi-path: Given the three previous relationships, an intention can be achieved by several combinations of sections. Such a topology is called a *multi-path*. In general, a map from its *Start* to its *Stop* intentions is a multi-path and contains multi-threads. For

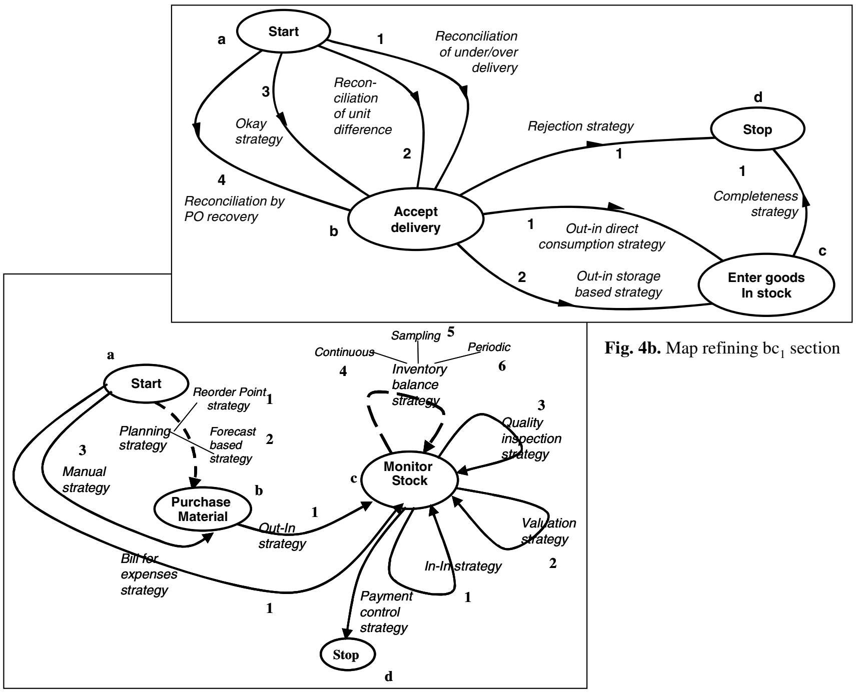

Fig. 4b. Map refining bc_1 section

Fig. 4a. The Material Management Map

Fig. 4. The Material Management Map (Fig. 4a) and the Map refining bc_1 section (Fig. 4b)

example, there is a *multi-path* to achieve the intention *Purchase Material*; either the path from *Start* to *Monitor Stock* via *Purchase Material* can be followed or the direct path from *Start* to *Monitor Stock* can be used.

Finally, it is possible to *refine* a section of a map into an entire map at a lower level of abstraction. For example, Fig. 4 shows the refinement of the section <*Purchase Material, Monitor Stock, Out-in strategy*> as a map (Fig. 4a). This refinement mechanism leads to model business intentionality as a hierarchy of maps.

3.2 Deriving Intentional Services from Maps

Having represented business intentionality as maps, we now proceed to determine services and their composition according to the *ISM*. We propose three key guidelines[1] to do this:

1- associate every section to an *atomic service*,
2- calculate all the *paths* of a map using an adaptation of the MacNaughton and Yamada's algorithm [12],
3- determine the aggregate services using the following *correspondences* between sections relationships in maps and service composition operators in *ISM* <path – composite>, <bundle – alternative>, <multi-thread – choice>, <multi-path – multi-path>. Since the entire map is, in general, a multi-path, it corresponds to an aggregate service.

We consider the three steps in turn and illustrate them with the MM map.

3.2.1 Associating Map Sections to Atomic Services

The first step consists of associating every section of a map to an atomic service. This correspondence leads in the case of the MM example, to services shown in Table1 in

Table 1. Services of the MM map

MM map sections	Intentional Services
ab_1	S *Purchase Material with reorder point strategy*
ab_2	S *Purchase Material with forecast strategy*
ab_3	S *Purchase Material Manually*
bc_1	S *Receive stock of purchased material*
ac_1	S *Receive stock by bill for expenses*
cc_1	S *Move stock*
cc_2	S *Evaluate value of stock*
cc_3	S *Inspect stock*
cc_4	S *Conduct Physical Inventory continuously*
cc_5	S *Conduct Physical Inventory by sampling*
cc_6	S *Conduct Physical Inventory periodically*
cd_1	S *Verify invoice against delivery*

[1] For sake of clarity, we deal here with guidelines for one single map whereas the entire process must deal with a hierarchy of maps. Rule 1 above needs then to be adapted (non refined sections are associated to atomic services) and an iteration step for every refined map shall be added.

correspondence with each of the 12 sections of the MM map. For sake of conciseness we use an abbreviated notation to refer to a section. We refer to each intention by a letter and to each strategy between a pair of intentions by a digit starting from 1 (see Fig. 4). Therefore, *ab3* is the reference of section <*Start, Monitor Stock, Manual strategy*> between the source intention *Start*, the target intention *Monitor Stock*, with the *Manual strategy* coded 3.

It can be seen that the name of each service reflects the business intention that can be achieved as well as the strategy to achieve it.

3.2.2 Calculating all Paths

We sketch an algorithm that automatically generates paths in the map and therefore, allow us to determine aggregate services as well as their nature, composite or variant. This algorithm is an adaptation of the MacNaughton and Yamada's algorithm [12] to calculate paths in a graph. This algorithm uses the different types of relationships between sections in a map that we introduced earlier.

The MacNaughton's algorithm is based on the two following formula:

Let *s* and *t* be the source and target intentions, *Q* the set of intermediary intentions including *s* and *t* and *P* the set of intermediate intentions excluding *s* and *t*.

The initial formula $Y_{s,Q,t}$ used to discover the set of all possible paths using the three operators that are the union ("∪"), the composition operator (".") and the iteration operator ("*") is:

$$Y_{s,\,Q,\,t} = \bullet\,(X^*_{s,\,Q\backslash\{s\},\,s},\ X_{\cdot s,\,Q\backslash\{s,\,t\},}\,t,\ X^*_{t,\,Q\backslash\{s,\,t\},\,t})$$

And given a particular intention *q* of *P*, the formula $X_{s,P,t}$ applied to discover the set of possible paths is:

$$X_{s,P,t} = \cup\,(X_{s,\,P\backslash\{q\},\,t}\,,\ \bullet(X_{s,\,P\backslash\{q\},\,q},\ X^*_{q,\,P\backslash\{q\},\,q},\ X_{q,\,P\backslash\{q\},\,t}))$$

We specialize the $X_{s,P,t}$ into paths, multi-paths, multi-threads and bundle relationships that we note as follows:

Bundle relationship between two intentions *k* and *l* is denoted $B_{kl} = \otimes(kl_1,\ kl_2\ \ldots kl_n)$ where the kl_i are the exclusive sections related by the bundle relationship. In Fig. 4, the bundle of planning strategies is $B_{ab} = \otimes(ab_1,\ ab_2)$.

Multi-thread relationship between two intentions *k* and *l* is denoted $MT_{kl} = \vee(kl_1\ kl_2,\ kl_n)$ where the kl_i are the sections related by the multi-thread relationship. Thus, the multi-thread between *Start* and *Purchase Material* in Fig. 4 is $MT_{ab} = \vee(B_{ab},\ ab_3)$.

Path relationship between two intentions *k* and *l* is denoted $P_{k,Q,l}$ where *Q* designates the set of intermediary intentions used to achieve the target intention *l* from the source intention *k*. A path relationship is based on the sequential composition operator "**.**" between sections and relationships of any kind. As an example, the path relationship in Fig. 4 between *Start* and *Monitor Stock* is denoted $P_{a,\{b\},c} = \bullet(\ MT_{ab},\ bc_1)$.

Multi-path relationship between two intentions *k* and *l* is denoted $MP_{k,Q,,l}$ where *Q* designates the set of intermediary intentions used to achieve the target intention *l* from the source one *k*. A multi-path relationship is based on the union operator "∪" between alternative paths. Thus, the multi-path in Fig. 4 between *Start* and *Stop* is denoted $MP_{a,\{b\},c} = \cup(ac_1,\ P_{a,\{b\},c\}})$.

The initial formula generating all the paths between the intentions a and d of Fig. 4 map, is:

$$Y_{a,\{a,b,c,d\},d} = \bullet(X^*_{a,\{b,c,d\},a}, X_{a,\{b,c\},d}, X^*_{d,\{b,c\},d})$$

The identified paths are summarized in Table 2.

Table 2. List of sections relationships

Type of relationship	Identified relationships
Path	$P_{a,\{b\},c} = \bullet\ (MT_{ab}, bc_1)$
	$P_{a,\{b,c\},d} = \bullet\ (MP_{a,\{b\},c}, MT_{cc}^*, cd_1)$
Multi-Path	$MP_{a,\{b\},c} = \cup\ (ac_1, P_{a,\{b\},c\}})$
Bundle	$B_{ab} = \otimes(ab_1, ab_2)$
	$B_{cc} = \otimes(cc_4, cc_5, cc_6)$
Multi-Thread	$MT_{ab} = \vee(B_{ab}, ab_3)$
	$MT_{cc} = \vee(cc_1, cc_2, cc_3, B_{cc})$

3.2.3 Determine Aggregate Services

Now, we establish a correspondence between section relationships in the map and aggregate service types. This correspondence is as follows: <path – *composite*>, <bundle – *alternative*>, <multi-thread- *choice*>, <multi-path- *multi-path*>. Table 3 presents the variant and composite services associated to the MM map. These are expressed with the set of variant and composite operators, namely ∨, ⊗, ∪, •, * introduced earlier in section 2.

Table 3. Components of the aggregate service S *Satisfy Material Need Efficiently*

Aggregate Types	Services
Variant services	$S_{\textit{Purchase Material Planning strategy}} = \otimes\ (S_{\textit{PM reorder point strategy}}, S_{\textit{PM with forecast strategy}})$
	$S_{\textit{Conduct Physical Inventory}} = \otimes\ (S_{\textit{CPI continuously}}, S_{\textit{CPI by periodically}}, S_{\textit{CPI by sampling}})$
	$S_{\textit{Purchase Material}} = \vee\ (S_{\textit{Purchase Material Manually}}, S_{\textit{Purchase Material Planning Strategy}})$
	$S_{\textit{Monitor Stock}} = \vee\ (S_{\textit{Conduct physical inventory}}, S_{\textit{Inspect stock}}, S_{\textit{Move stock}}, S_{\textit{Evaluate value of stock}})$
	$S_{\textit{Receive stock}} = \cup\ (S_{\textit{Receive stock by bill for expenses}}, S_{\textit{Receive stock normally}})$
Composite services	$S_{\textit{Satisfy Material Need Efficiently}} = \bullet\ (S_{\textit{Receive stock}}, S_{\textit{Monitor Stock}}^*, S_{\textit{Verify invoice against delivery}})$
	$S_{\textit{Receive stock normally}} = \bullet\ (S_{\textit{Purchase material}}, S_{\textit{Receive stock of purchased material}})$

It is to be noted that the entire MM map is associated to a composite service S *Satisfy Material Need Efficiently* having the intention *Satisfy Material Need Efficiently.* This aggregate service is a composition of three services, S *Receive stock,* S **Monitor* Stock, and S *Verify invoice against delivery*. The first one of these is a multi-path with the intention to *Receive Material in stock.* The second is a set of variant services to achieve the intention *Monitor Stock.* The third one is an atomic service intended to *Verify invoices against delivery.*

4 Adapting Services

Since an aggregate service captures a full range of variants to achieve the root service intention, when the business agent (Fig. 2) desires to use the service he has selected there is an adaptation issue. The issue of adaptation is that of determining which variant services and which combination of variant services are relevant to the situation at hand.

We again believe that adaptation must be driven by business intentions and identified two different ways in which adaptation can be done:

- *Design time adaptation* permits a selection of a combination of variants that might result in only one composite service; i.e. one path from Start to Stop in the map.
- *Run time adaptation* allows to leave a large degree of variability in the adapted aggregate service and the desired variant services can then be selected *dynamically* at enactment time.

This section describes how the different combinations of services in an *ISM* aggregate service can be mapped to an agent architecture that monitors the navigation through service relationships and thus allows dynamic service selection at run time.

4.1 The Agent Architecture

In order to monitor the navigation among the composition of services and offer to the business agent the choice of variants he/she wants to execute, we build a hierarchy of agents to managing service relationships and handing over the execution of atomic services. The hierarchy is composed of two kinds of agents: control and executor agents.

- *An executor agent* is a self-contained unit that implements an atomic service; this can be done by handing over the control to a traditional service composition engine such as the BPEL4WS engine [13].
- *A control agent* controls the selection and execution of a given composition, i.e. a path in a map (executors or/and other control agents). We distinguish four kinds of control agents for each of the four operators "$\cup$" (*multi-path),* **"."** (*path),* "$\vee$" (*thread)* and "$\otimes$" (*bundle)* control agents. They respectively control the selection and execution of the paths related by multi-path, path, multi-thread and bundle relationships.

In order to build the hierarchy, we defined mapping rules that are briefly sketched in the following. We first introduce one executor for each atomic service. As can be seen in Fig. 5, there is a one to one correspondence between atomic services and executors. For example, the service ab_1 is mapped to an executor having the same name. Executor agents are the leaves of the hierarchy.

Higher levels correspond to control agents. There is a kind of control agent for each kind of service relationship. For example, in Fig. 5, the multi-thread relationship MT_{ab} (see Table 2) is associated to a multi-thread control agent having the same name. We first identify control agents using a one-to-one correspondence and then, make some simplifications, for example, a path relationship composed of one atomic service in not mapped to a control agent.

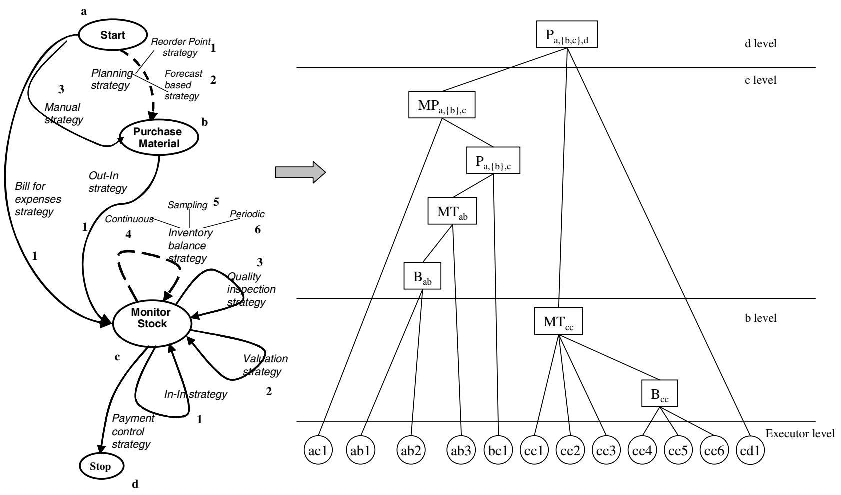

Fig. 5. Agent architecture

Control agents are organized at different levels. Each level is responsible for the achievement of a service intention. The top level of the hierarchy corresponds to the *Stop* intention and is responsible of the achievement of the goal of the whole aggregate service (map). The next level is related to the intention preceding the *Stop* intention. The bottom level of the hierarchy is composed of the executor agents. The hierarchy of Fig. 5 is composed of four levels related to the intentions of the MM map. The control agents of each level control children agents belonging to the same level or to the levels below.

4.2 Service Agent Support

Clearly, the ISOA departs from the usual SOA binding mechanism in providing an enactment mechanism that permits dynamic selection of services at run time. This is compatible with the business-oriented view of the *ISM* and the need for business agents to adapt decision making ‘on the fly’. We believe that it is possible for business people to perform this adaptation. This is because knowledge of the business characteristics and an analysis based on these is enough to make the adaptation decision.

5 Related Work

Generally speaking, research on service description, composition and adaptation is relevant for our work [3].

Typical descriptions of services are based on finite state formalisms, e.g., in [14] [15] services are represented as state charts, in [16] services are modeled as Mealy machines and in [17], services are represented as finite state machines. The *ISM* shares with these approaches the need to describe service to ease their retrieval but departs from their *function driven* perspective to propose an *intention drive* of service description. As a consequence, *ISM* service descriptions will bring out the business intention that the service allows to fulfill and pre and post conditions instead of defining the signatures of operations that can be invoked on class objects. We believe that this contribute to avoid the current mismatch of languages between low level services descriptions such as WSDL statements and business perceived services.

Our description of intentional services has some similarities with semantic descriptions as found in [18][19][20]. Annotations which provide these semantic descriptions are compared to ontology elements in order to enrich usual retrieval mechanism. However, none of these semantic descriptions seem to be based on goal matching.

Our approach borrows from goal driven approaches in Requirements Engineering [21][22] the idea of goal decomposition and goal refinement through AND/OR graphs. This leads to an intention driven service composition: an *ISM* aggregate service has a high level, strategic intention as its key characteristic and its composition is reflecting the intention decomposition into sub-intentions that can be themselves fulfilled thanks to a composition of lower level sub-intentions etc. till operational intentions related to atomic services are found. By contrast most proposals are based on the idea of flow-composed services in which services are black boxes exchanging input/output parameters [23][17][24].

A large body of research work [25][26][14][15][16] deals with service execution: (i) the peer-to-peer architecture in which the individual service interact among themselves and with the client directly, and (ii) the mediated architecture in which the control over the available services is centralized. Our approach fits best to the peer-to-peer perspective but needs specific mechanism to cope with the adaptation issue.

From a methodological viewpoint, our proposal is close to [27] as both share the idea to capture service needs from exploring business goals. In [27] a revised *Tropos* design process is used to support service discovery and composition by offering a roadmap that relates stakeholder goals to collections of services available in different directories.

6 Conclusion

In this paper we introduced the notion of intentional service as one described in terms of the business goal it allows to fulfill. We also showed that ISOA service composition is intention driven and reflects business needs. This is in accordance with our view that business executives must be provided with a description of services available in a service portfolio that is adapted to their own perceived needs.

The paper considered in some detail the three roles of our ISOA architecture:

- E-business provider, who looks at a business, identifies its intentions, derives and publishes services in the intentional service registry.
- Intentional service registry where services are available. The descriptors of services and the typology of services being kept are modeled in the *ISM*.
- Business agent who retrieves services from the registry and dynamically navigates through aggregate services composition graphs using the agent architecture. The appropriate aggregate variant is thus available for execution.

Whereas the three roles of ISOA correspond to the service provider, registry and client roles of the SOA, it is to be noted that ISOA services, aside from supporting business intentions, are also more complex than SOA ones. This is because of aggregate variants that provide flexibility to the business agent in performing the task at hand. In contrast SOA services are fixed and are available on a 'take it or leave it' basis.

The proposed approach is still work in progress. Current research aims at developing (a) an intention driven search mechanism for the selection of services on the basis of the business goal they allow to fulfil and (b) a software tool to guide the discovery of aggregate service through maps.

References

1. Papazoglou, M-P., Giunchiglia, F., Kraemer, B., Traverso, P.: Service Oriented Computing Network, The new computing paradigm for the network world (2003)
2. Papazoglou, M-P.: Service-Oriented Computing: Concepts, Characteristics and Directions, WISE'03, Rome, Italy (2003)
3. Alonso, G., Casati, F., Kuno, H., Machiraju, V.: Web Services. In: Concepts, Architectures and Applications, Springer, Heidelberg (2004)
4. Papazoglou, M-P., Georgakopoulos, D.: Service-Oriented Computing. Communication of the ACM, 46(10) (2003)
5. W3C Web Service Description Language (WSDL) Version 1.2. W3C Working Draft 3, (2003) http://www.w3.org/TR/wsdl12/
6. Piccinelli, G., Emmerich, W., Williams, S-L., Stearns, M.: A Model-Driven Architecture for Electronic Service Management Systems. In: Orlowska, M.E., Weerawarana, S., Papazoglou, M.M.P., Yang, J. (eds.) ICSOC 2003. LNCS, vol. 2910, pp. 241–255. Springer, Heidelberg (2003)
7. Arsanjani, A.: Service-oriented modelling and architecture. (November 2004) http://www-128.ibm.com/developerworks/webservices/library/ws-soa-design1/
8. Zimmermann, O., Krogdahl, P., Gee, C.: Elements of Service-Oriented Analysis and Design (2004)
   http://www-128.ibm.com/developmentworks/webservices/library/ws-soad1/
9. Jackson, M.: Software Requirements and Specifications. In: A lexicon of practice, principles and prejudices, p. 256. Addison-Wesley, New York (August 1995)
10. Rolland, C., Souveyet, C., Ben Achour, C.: Guiding Goal Modelling using Scenarios. IEEE Transactions on Software Engineering, Special Issue on Scenario Management 24(12), 1055–1071 (1998)
11. Rolland, C., Prakash, N.: Bridging the gap between Organizational needs and ERP functionality. Requirement Engineering Journal (2000)
12. MacNaughton, R.: Yamada: Regular expressions and state graphs for automata. IEEE transactions on electronic computers EC-9, 39–47 (1960)
13. Andrews, T., Curbera, F., Dholakia, H.: Microsoft, IBM, and SAP. BPEL4WS version 1.1, (2003) http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/
14. Mecella, M., Pernici, B., Craca, P.: Compatibility of e-Services in a Cooperative Multi-Platform Environment. In: Proc. VLDB-TES (2001)
15. Fauvet, M-C., Dumas, M., Benatallah, B., Paik, H.: Peer-to-Peer Traced Execution of Composite Services.In: Proc. of VLDB-TES (2001)

16. Bultan, T., Fu, X., Hull, R., Su, J.: Conversation Specification : A New Approach to design and analysis of E-Service Composition. In: Proc. of the WWW'03 Conference (2003)
17. Berardi, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Mecella, M.: Automatic Composition of e-Services that Export their Behavior. In: Proc. of WES 2003 (2003)
18. DAML-S Defense Advanced Research Projects Agency: DARPA agents markup language- Services (DAML-S). http://www. Daml.org/services/
19. Sirin, E., Parsia, B.: Planning for semantic web services in Semantic web services workshop at ISWC'04. (2004) http://www.mindswap.org/papers/SWS-ISWC04.pdf
20. Horrocks, I., van Harmelen, F., Patel-Schneider, P., Berners-Lee, T., Brickley, D., Connoly, D., Dean, M., Decker, S., Fensel, D., Hayes, P., Heflin, J., Hendler, J., Lassila, O., McGuinness, D., Stein, L.A.: DAML+OIL (2001) http://www. Daml.org/2001/03/daml+oil-index.html
21. Van Lamsweerde, A., Dairmont, R., Massonet, P.: Goal Directed Elaboration of Requirements for a Meeting Scheduler: Problems and Lessons Learnt. In: Proc. Of RE'95, pp. 194–204. IEEE, New York (1995)
22. Yu, E.: Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering. In: Proceedings of the 3rd IEEE Int. Symp. On RE'97. pp. 226-235 Washington D.C., USA (January 6-8 1997)
23. Yang, J., Papazoglou, M-P.: Service Components for Managing the Life-Cycle of Service Compositions. Information Systems Journal (2003)
24. Dijkman, R-M.: A Basic Design Model for Service-Oriented Design, ArCo/WP1/T1/D2/V1.00, (2003)
25. Casati, F., Shan, M.: Dynamic and Adaptive Composition of e-Services, Information Systems, 6(3) (2001)
26. McIltraith, S., Son, T., Zeng, H.: Semantic web services, IEEE Intelligent Systems, 16(2) (2001)
27. Perini, A., Susi, A., Mylopoulos, J.: Tropos Design Process for Web Services, 1st Int. Workshop on SOC: Consequences for Engineering Requirements, Paris (2005)