Management Services – A Framework for Design

Hans Weigand[1], Paul Johannesson[2], Birger Andersson[2],
Jeewanie Jayasinghe Arachchige[1], and Maria Bergholtz[2]

[1] Tilburg University, P.O.Box 90153,
5000 LE Tilburg, The Netherlands
{H.Weigand,J.JayasingheArachchig}@uvt.nl
[2] Royal Institute of Technology
Department of Computer and Systems Sciences, Sweden
{pajo,ba,maria}@dsv.su.se

Abstract. The Service-Oriented Architecture has rapidly become the de facto standard for modern information systems. Although recently considerable research attention has been paid to the management of services, several gaps can still be observed. Service management as far as it is automated is either mixed up with the operational service logic itself, or handled in a separate not service-oriented system, such as a BAM platform. In addition, there is a growing business demand for value-driven service management. In this paper, a general framework for management service design is presented that covers both business services and software services and is rooted in the business ontology REA, extended with a REA management ontology. The framework is applied to two different case studies, one in the Italian wine industry and one related to a robot cleaner.

Keywords: service design, REA, autonomic computing, management control.

1 Introduction

The Service-Oriented Architecture (SOA) has rapidly become the de facto standard for modern information systems. Having started with a focus on service description and discovery, SOA research shifted its attention to service composition in the second phase. The next phase, according to [13] in 2007, would focus on service management defined as "the control and monitoring of SOA-based applications throughout their lifecycle". The most prominent functions of service management include SLA management, auditing, monitoring and troubleshooting, dynamic resource provisioning, service lifecycle management (e.g. versioning) and scalability/extensibility. However, although considerable work has been done on these topics in the last few years, results so far are fragmented and limited. Early standards related to service management (MUWS/MOWS; see oasis.org) have become obsolete.

Several research gaps can be observed. Service management as far as it is automated is often mixed up with the operational service logic itself, or it is handled in a separate not service-oriented system, such as a BAM (Business Activity Monitoring) platform. In addition, service management, including business process management, is still mainly focused on execution correctness [1], whereas there is a growing

H. Mouratidis and C. Rolland (Eds.): CAiSE 2011, LNCS 6741, pp. 582–596, 2011.

business demand for value-driven service management. This also requires a better integration of software services and business services [10] – which still is the vision of service science anyway.

Along with SOA, the last decade has witnessed increased research efforts on self-adaptive or autonomic software. We refer to [20, 4] for recent overviews. Self-adaptive software embodies a closed-loop control mechanism that includes sensors and effectors, linked through processes of monitoring, detecting, deciding and acting. Despite considerable progress in sub areas, there are also still many challenges, including the question on how to integrate self-adaptation functionality in the SOA architecture. Is it possible to provide Management as a Service?

The control loop is taken by Forrester analyst Bartels as the backbone for Smart Computing [1]. Smart Computing is supposed to be the challenge of the coming decade and integrates the following technologies: Awareness (sensors, RFID chips, video cameras), Analysis (business intelligence, process mining), Alternatives (rule engines, workflow systems) and Actions (leveraging existing products), with Auditability as an overall concern. The big business challenge lies in optimizing the value of and the return on assets and minimizing the costs and risks from liabilities by far better *real-time* awareness of their status. Assets include both physical resources such as buildings and trucks, but also intangible ones such as software, or brand.

The research objective of this paper is to develop a general framework for management service design that covers both business services (as in Smart Computing) and software services (as in Autonomic Computing). The framework must be value-driven and truly service-oriented. To arrive at rigorous and relevant research results, we use Peffers' design science phases [14]. The *problem identification and motivation* is stated in section 1 and 2. Our *solution objective* is to develop an integrated framework for value-driven service management. In section 3 we lay a formal foundation by extending the REA business ontology with a REA management ontology and introduce a general framework of services that is applied recursively to management services (*design and development*). The framework is used in two case studies (*demonstration*). The first case was developed in the context of the S-Cube project [17] and concerns an Italian wine industry. The advantage of using this case study is the comparison it allows with other approaches. To explore the applicability range of our approach, the second case is about a robot cleaner.

2 Related Research

Given limited space, we can only present a brief overview of all the relevant fields.

The vision of Autonomic Computing was presented by Kephart and Chess in 2003 [7], and recently evaluated in [4]. The evaluation observes that the vision has been broadened to "the application of advanced technology to the management of advanced technology" and as such is still highly relevant. A notable omission in the original vision was the communication component, and research has been devoted to developing a so-called *knowledge plane*. What is also still lacking is an understanding of the broader software engineering aspects of autonomic system development. This includes such basic questions as when a system can be said to be "correct" if its behavior is expected to change over time. [4] also pleads for a comprehensive systems

theory for adaptive systems that allows to reason not only about the "what" of monitoring and adaptation but also about the "how". Finally, it claims that current solutions are too much focused on isolated problems, and that we need an integrated autonomic systems engineering approach to avoid undesirable feature interaction.

Within the software engineering domain, runtime adaptation has become a highly relevant research topic. In this context, Model-Driven development and Software Product Lines modeling techniques are used and extended to Dynamic Software Product Lines that include variability transformations [3, 26, 8]. This work confirms the research gap identified by [4] that the "how" of the adaptation is no longer something that can be abstracted from.

Another line of research is considering modeling and monitoring requirements for adaptive systems. Goal-based approaches, such as Tropos [2] have been applied to the specification of requirements of self-adaptive systems. Goal-based models are well suited to exploring high-level requirements and it is natural to use goal models to represent alternative behaviors that are possible when the environment changes. Goal-based models have a natural fit with agent-oriented implementation platforms [11].

In the management literature, system theory and control theory have been around for a long time. In the standard management control textbook of Simons [19], four types of "control systems" are distinguished: diagnostic control systems, interactive control systems, belief systems and boundary systems. *Diagnostic control* systems correspond to the traditional cybernetic approach and are aimed at controlling results using a closed control cycle. This mode of control is important but it also has its limitations, according to Simons. *Belief systems* express norms and core values in the organization that are aimed at controlling (or influencing) the value systems of the people involved. *Boundary systems* constrain the behavior of the organization in the face of risks to be avoided. *Interactive control* is focused on handling uncertainties and on "doing the right thing", rather than doing the things right, as in diagnostic control, and is typically realized in the form of interaction.

3 A Framework for Management Service Design

3.1 The REA Business Ontology

The Resource-Event-Agent (REA) ontology was first formulated in [9] and has been developed further, e.g. in [21,5,6]. The following is a short overview of the core concepts of the REA ontology based on [23].

A *resource* is any object that is under the control of an agent and regarded as valuable by some agent. This includes goods and services. Resources are modified or exchanged in processes. A *conversion process* uses some input resources to produce new or modify existing resources, like in manufacturing. An *exchange process* occurs as two agents exchange (provide, receive) resources. To acquire a resource an agent has to give up some other resource. An *agent* is an individual or organization capable of having control over economic resources, and transferring or receiving the control to or from other agents. The constituents of processes are called *economic events*. REA recognizes two kinds of duality axioms between events: conversion duality and exchange duality.

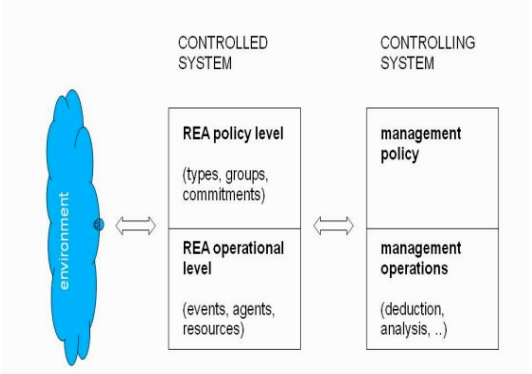

Fig. 1. REA levels of controlled enterprise system, derived from [22]

The event records give an answer to the question "what has happened?", but not to the question "what is planned or scheduled – what *should* happen?" In REA this is modeled at the policy level. This level allows talking about types and groups as well as commitments. Although no specific details for the structure of these types and commitments are given, examples in [5] show how REA does handle schedules, plans and integrity constraints in this way. [22] gives an enterprise architecture based on REA that makes a distinction between controlled system and controlling system (Fig. 1). The controlling system interacts with both levels of the controlled system, monitoring the facts at the operational level and changing the plans, standards, schedules etc at the policy level. As explained by [22], the controlling system is also a subsystem of the enterprise, so it does also have an operational and policy level.

3.2 REA Management Ontology

The business ontology of REA is strong in formalizing the operational level of the enterprise on an abstract economic level. However, the later extensions on the policy level have not been worked out as thoroughly. The occurrence of general abstractions (types, groups) on the one hand and commitments on the other hand in the same basket "policy level" is not really satisfactory. The dynamics of this policy level, e.g. how commitments are created and resolved, are also not covered, although this has a clear business impact and cannot be left out of scope. The work of [22] adds a useful distinction between controlled system and controlling system, but it does not spell out what exactly is processed in the controlling system when it talks about deduction, analysis, etc. Therefore we have developed an extension of REA ("management ontology") that we will use as a basis for our design framework (Fig.2).

The lower left corner summarizes the REA concepts of resource, event and agent (at operational level) as well as types and groups (of resources, events and agents). We group them together under the category *REA referent*. REA referents are "referred" to in *intentional resources*. Three categories of intentional resources are distinguished on the basis of what Searle [18] has called direction-of-fit: *assertives*, *directives* and *evaluatives*. An (basic) assertive says that some REA event has occurred, whereas a directive says that some REA event should occur. Directives are defined as having a world-to-word fit and as such are a generalization of commitments that were already in the REA business ontology, as part of the policy level.

However, unlike types and groups, commitments do not correspond to an abstraction; like assertives, they have a propositional structure with references to REA referents. We have extended REA also with a value dimension. Values qualify REA referents, in terms of good, bad, satisfactory, fast etc. *Evaluations* are intentional resources that are produced using other intentional resources and assign values; they do not have a direction-of-fit, like the expressive in [18].

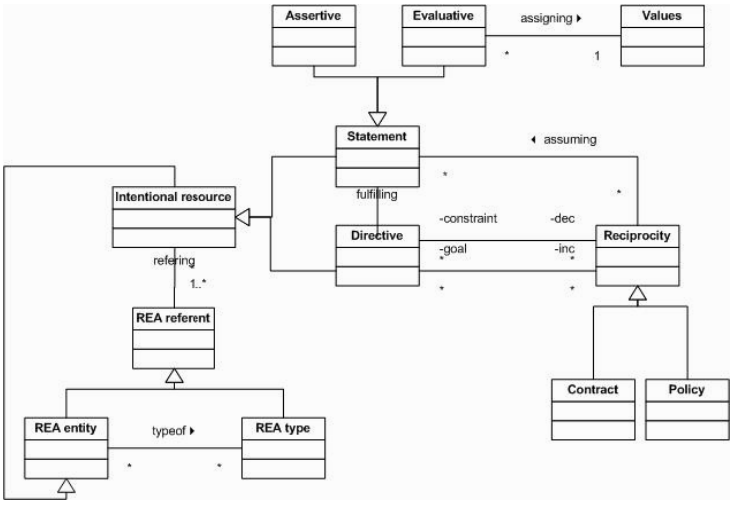

Fig. 2. REA management ontology (UML style)

As any resources, intentional resources are processed by events that we call *intentional events*. For instance, commitments are created or removed by commit and decommit events [24]. Intentional events are typically realized by means of communicative acts to be represented on the process level.

In REA, commitments are grouped together in contracts. A *contract* contains reciprocal commitments (what the agent will give versus what he aims to receive) and may contain additional terms. From a management perspective, *policies* are important concepts as well. According to the BusinessDictionary.com, a policy is "*a set of basic principles and associated guidelines, formulated and enforced by the governing body of an organization, to direct and limit its actions in pursuit of long-term goals*". We have formalized them in analogy to contracts as a group of intentional resources obeying the reciprocity principle: what the agent gives in versus what he aims to achieve. *Constraints* are what the agent gives in (directives that limit the actions of the controlled system) and *goals* are what the agent wants to gain in return (directives to be fulfilled by monitored assertives or evaluations). In addition, the policy may contain *assumptions* in the form of testable assertives.

The definition of policy can be seen as a generalization of the policy pattern in [6], which defines a policy entity as something that encapsulates constraints on economic exchanges and conversions and that "applies" to a group. In our definition, constraints are part of the policy, but always linked via the policy to goals. In our view, the "apply" relationship is a special kind of "referring".

Policies can be defined on two levels, the "what" and the "how". This abstraction is important, since it allows for adaptation of the controlled system, within given boundaries. The way the abstract policy is enforced may depend on the (monitored) situation and on design choices made by the controlling system [25]. In the following, we will use the term "specification" for the enforced (and executable) policy. How far the policy abstracts from the specification is a design choice that will differ from one case to the other.

A *management process* is like a REA economic process containing both conversion events and exchange events. For instance, an inference service *uses* certain assertives and some rule base (formally, these are also assertive) in order to *produce* other assertives. Management discourse is based on an exchange duality balancing incoming assertives ("subscribe") with outgoing ones ("publish").

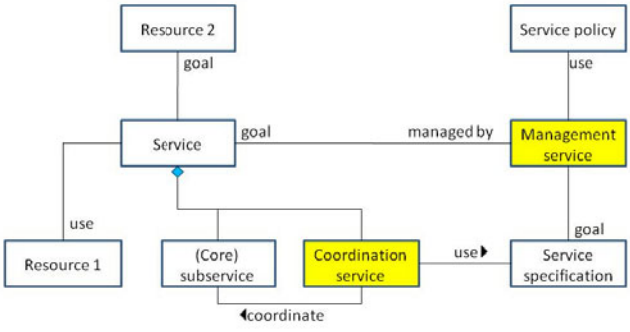

Fig. 3. Generic service model

3.3 Generic Service Model

Our generic service model is based on [23] and illustrated informally in Fig. 3. A service *adds value* to some *resource*, called its goal, consuming or making use of other resources. Including the resources in the service model is in line with REA and has been argued for independently by [16]. A service can be regarded as a small service system and decomposed into *subservices*. *Core subservices* contribute to the service goal, whereas *coordination services* are subservices that coordinate – manage the dependencies between – core subservices [24], using a *specification* derived from the *policy*. We model subservices with a part-of relationship, but in terms of the REA ontology, they are related to the super service by a use relationship just like resources. *Management services* have been defined in [23] as a kind of enhancing services that adds value to a service. They are not subservices of the service in question, with an active role in the operational process, but manipulate the (operational) service by enforcing one or more service policies.

The service model is intended to be generic. It can be applied to business services, such as a transport, or a hair-dressing, but also to a software service. Let us illustrate the transport case: the resources that are used or consumed include the lorry, the petrol and the driver's labor hours, each of them having a certain value. The transport

adds value to the transported goods. This value is reflected in the price that the customer is willing to pay for the transport service. Transport *management* is an extensive task that includes capacity planning and scheduling as well as actual route planning. The transport itself can be decomposed into subservices such as driving, unloading and a coordination service for selecting a route. A policy may contain the constraint that the lorry should select the cheapest route in terms of fuel consumption.

The software case can be illustrated as follows. The same transport company may have a tracking & tracing service for customers. The resources used by this software service are informational in nature: in particular, they include the GPS data about the lorries. The service adds value to the transport (the transport service has a higher value for the customer when he can track his cargo – a case of "information enrichment" [12]). For its implementation this web service makes use of utility or infrastructural services based on resources such as CPU time, data storage and the network. Once the service is in place, there may be a management service to support it. For instance, the management service may monitor and adapt the service interface in order to maintain interoperability. Note that management services may be automated or semi-automated independently of the automation of the service itself.

The *value of a service* in a certain period of time is the difference between the total value increase on the one side and the total decrease at the other side. In order to realize value-based service management, it is therefore necessary that each resource is valued. There is a long tradition of cost accounting that we can rely on to implement this requirement, the details of which are not in the scope of this paper. What is important for keeping the valuation consistent is to integrate all services into the *value cycle* of the company, denoted as "cash wheel" in [19]. As traditional audit theory teaches, each company has a value cycle consisting of sales processes that generate money, consuming resources, and purchasing processes that acquire resources spending money. Depending on the type of company – sales, manufacturing, etc. – the value cycle can be drawn a bit more precise, but the structure is always the same. The *value of the value cycle* is the profit that is generated over a certain period of time minus the investments that have been added. Each service is directly or indirectly included in the value cycle, akin to Porter's distinction between primary and support activities of the value chain [15]. Directly included are services such as manufacturing, sales and purchasing. Indirectly included are for instance management services that contribute to primary services or other enhancing services. At the end of the day, the valuations of all services should be consistent, that is, the sum of these valuations should be equal to the value of the value cycle (for a certain period). We regard this principle not only important from an accounting point of view, but also as a useful constraint for business modeling.

3.4 Management as a Service

We have conceptualized management as an enhancing service. In Software Engineering, the idea of separating operational and management concerns is not new. In the field of self-adaptive software, an equivalent distinction is made between internal and external adaptation [20]. Internal approaches intertwine application and adaptation logic. This has certain drawbacks. External approaches use an external adaptation engine or manager that contains the adaptation logic, the other part being called the

"adaptable software". By conceptualizing management as a service, we follow an external approach. Note that this is not a formal necessity but an architectural choice.

We propose a *fractal* design approach in the sense that the same generic service model is applied to management services. So a management service is itself something that uses resources. These include operational resources, e.g. labour hours, and intentional resources. The management service may have subservices, such as a monitoring service, and a management subservice may have a manager service. This service-oriented approach increases reusability. The advantage of the fractal approach is that it makes the design completely service-oriented, not only its operational part. This is in contrast to other approaches that for instance conceptualise BDI agents as management services, or monolithic BAM software.

What does a management service actually do? Most current approaches in the field of self-adaptive software follow classical control theory and posit a control loop, also called MAPE cycle (Monitor-Analyze-Plan-Execute). The same cycle underlies the deliberation cycle that is used in multi-agent systems (with Beliefs, Desires and Intentions). Our approach is required to be business-driven and service-oriented. Following the management control literature ([19]) we call it the diagnostic control cycle.

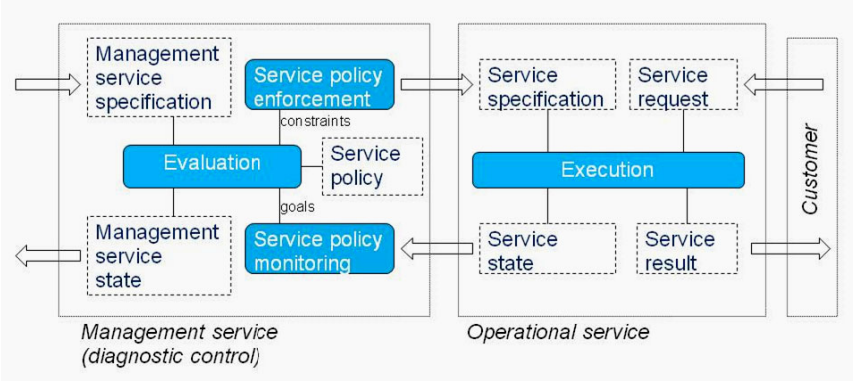

Fig. 4. Diagnostic control cycle together with the service interaction cycle. Dashed rectangles indicate intentional resources, coloured boxes indicate services

Fig. 4 depicts the generic service-oriented management architecture for the diagnostic control cycle. On the right hand side, we see the traditional service interaction cycle: the customer sends a request to the service provider. The execution produces, perhaps iteratively, a certain state that corresponds to and so fulfills the request, and this result is returned to the customer for evaluation. However, the execution does more than that. From a management perspective, the execution is the realization of the service specification. So there is another interaction cycle, between management service and operational service: the manager enforces a *service policy* on the operational service. In the case of software services, the *service specification* may take the form of a BPEL specification, or a set of business rules (cf. [8, 26] for how to implement policy enforcement based on such models). The execution produces a certain state (set of *assertive* – this reporting is also governed by policy constraints). The state information is returned to the manager, where it is typically aggregated by monitoring

services and then evaluated. If the evaluation is not satisfactory (does not match the policy *goals*), the service specification is adapted. The policy will usually contain conditional constraints that become effective in the case of contingencies, akin to terms in a contract or a mitigation plan.

It is possible that the operational service policy has to evolve itself. However, this is not the responsibility of the management service. If we want this type of self-adaptation, a second management level has to be introduced, in accordance with our fractal design principle. In that case, the Management Service specification in Fig. 4 is not fixed but itself the result of a Management Policy enforcement process.

Three kinds of management subservices can be distinguished. A *monitoring* service uses and produces assertives. An *evaluation* service uses assertives and produces evaluatives. An *enforcement* service enforces policies, using evaluations and possibly assertives and producing directives. Further specializations are, for instance, sensor services, aggregation services, inference services and data transformation services as specific monitoring services.

The flexibility within the service policy that enables varying enforcements can be realized in several ways that go beyond the scope of this paper. We just want to mention two options. The policy may consist of a fixed set of alternatives, as in the variability transformations approach [3], that are selected on the basis of the evaluation. Or the policy contains a parameter whose value is dependent on the evaluation. An example is "credit level" in an order processing service. If the credit level is too low, the company losses because of non-payments. If the credit level is too high, the company misses sales opportunities. To find an optimal credit level and adapt if when the circumstances change the manager can (re-)calculate the parameter value by means of a stochastic optimization algorithm.

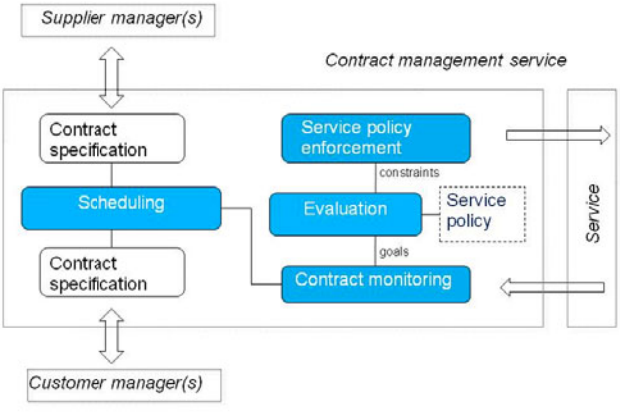

Fig. 5. Contract management cycle

According to [19], diagnostic control is the "automated pilot" that allows the human manager to spend his time on other things, in particular interactive control. The way Simons presents this it is not a homogeneous group. An important subclass concerns interactions with service stakeholders about the service requirements. These

requirements can be diverse, but include requirements on future capacity. These requirements are passed through the value chain in reverse direction, from customer (market demands) to sales and further on to production and procurement. To account for this kind of interactive control, we need another management cycle (Fig. 5).

The manager interacts with other managers in the value chain at customer and supplier side. The customer manager's requests do not concern a particular service instance, but a certain state or quality of the service as such. For instance, that the service has a certain capacity at a specified time. This leads to certain *commitments* that are part of a REA *contract*. The synchronization of contracts is what is called scheduling. The purpose of a schedule is to make sure that for all services the needed resources are identified, as well as when they will be needed [6:108]. A schedule is a collection of increment and decrement commitments, as well as mitigation plans. The increment commitments indicate the availability of the service at some future time, or the availability of the resource produced by the service. Decrement commitments concern the resources (subservices or resources produced by subservices) needed to fulfill the increment commitments. These decrement commitments must be gained from the managers of the supplying subservices. In our conceptualization, the schedule is not a separate entity but the combination of these contractual commitments. Note that the scheduling usually runs independently from the operational service. It only prevents the operational service to break down when the actual service requests come. However, the scheduling may influence the operational service. For instance, if the subservice providers are not able to commit to the required resource capacity, this is forwarded as such via the contract monitoring, so that the service policy enforcement can pro-actively find and bind other suppliers.

A second important subclass of interactive control distinguished by Simons is the ongoing conversations on probing the assumptions underlying the diagnostic control settings. One of the manager's interactive control tasks is to adapt the service policy when its assumptions do not hold anymore or to anticipate such a break-down. This can be realized by a *discourse* between managers, akin to the above-mentioned knowledge plane [4].

As mentioned earlier, more control systems could be distinguished – boundary systems and belief systems. Whether these can be realized as special cases of the other ones, or deserve to be identified independently, is a question for future research.

3.5 Design Method

Following the fractal approach, a comprehensive design method for management services (we ignore other aspects here) looks roughly as follows:

Step 1 Identify core business services
Step 2 Identify coordination and management services per core service
Step 3 Identify management subservices per management service
Step 4 Identify software services that may support any core service, coordination service, or management (sub) service
Step 5 Identify software services that manage the software services from Step 4 as well as subservices of these management services

The first step is general. To identify core business services, it is recommendable to use the value cycle of the enterprise as a reference. In step 2, management services are

identified. Whether this is feasible for all services in the enterprise depends on their *maturity level*. Some enterprises will require this maturity only for services that are of strategic importance. For the selected management services, a decomposition step is made in which subservices are modeled, related to the three different management control types distinguished above. Step 4 is making the match between required services and available IT support. This IT support can range from traditional MIS support software to business intelligence tools and to ubiquitous computing tools such as smart sensors. Each of the tools must get a service interface. At this point, the method can be complemented by existing industrial service engineering methods, as long as the architectural distinction between operational and management service is maintained. In step 5, we repeat step 2-4 of determining management services, but now for the software services. Some of these management services will be fully automated (autonomic computing), other semi-automated.

4 Demonstration

4.1 BSRM Modeling Notation

We use a simple modeling notation to model the services called BSRM (Business Service and Resource Modeling – not published yet). For the clear differentiation between services and the other resources we use different symbols. As far as terminology is concerned, we avoid adding the word "service" to each service name. A summary of the modeling notation is as follows:

- Services are denoted as rounded rectangle
  - exchange services are denoted as rounded rectangles with "exchange" label
  - conversion services are denoted as colored rounded rectangles
- Physical resources are denoted as rectangles
- Intentional resource are denoted as dashed rectangles
- PartOf relationship is denoted as a line with diamond end
- Management relationship is denoted as solid arrow with "+" label
- Stockflow relationship :
  - Inflow : arrow pointing to the resource/service
  - Outflow : arrow pointing from the resource/service

4.2 Italian Wine Producing

The proposed management service model has been evaluated in a real world case study of wine production [17]. According to the case description, the goal of the Wine Producer is to maximize his production in order to adapt the monitored market needs. During the wine producing process quality assurance plays a major role. The Quality Manager, the Agronomist who is an expert of a branch of agriculture which deals with field-crop production and soil management, and the Oenologist who is an expert in wine and wine production involved in this process. They have to observe the vineyard parameters and to react to critical conditions that may happen during the cultivation

phase. The wine production case has major phases namely vineyard cultivation, harvesting, fermentation and wine distribution and selling.

Following the management service modeling approach the first step is modeling core business services at the operational level (Fig.6).

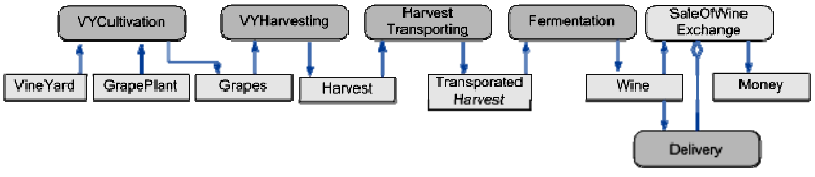

Fig. 6. BSRM model for wine production (core services operational layer)

The core services of the wine production are vineyard cultivation, vineyard harvesting, harvest transporting, fermentation, and sales of wine. There is one exchange service which is sales of wine and all the others are conversion services. Vineyard cultivation is the first step of the wine production process. It is a conversion service and it uses resources Grape plants and Vineyard and produces Grapes. The next conversion service VYHarvesting uses the Grapes and produces Harvest. The rest of the core services use and produce resources are as depicted in Fig. 6. Sales of wine which is an exchange service generates money in return for selling wine. To close the value cycle, the money derived from the sales of wine is spent on different activities in the wine producing process, for example to purchase grape plants, but we did not include it here because of space limitations.

Next, we take the step 2 and 3 from 3.5 together, focusing on the vineyard cultivation core service (Fig. 7). We look for management services corresponding to the three control cycles. Vineyard cultivation management service *VYCultivationMgt* is a *contract management service* responsible for the cultivation process and the VY activity planning and labor allocation are subservices to this management service. VY activity planning builds on contracts set up between *VYCultivation* and Sales, indirectly based on market information. VY quality management (*VYQualityMgt*.) service is a *diagnostic control service*. It is possible to identify two subservices to VYQualityMgt, namely VY activity monitoring and recovery management. It uses a number of intentional resources as input and produces a service *policy* in the form of a recovery action list. Climatic data is an example of assertives that are defined as *policy assumptions*. To acquire these assertives, the management service presumably relies on a *discourse* (not included in the model). VY parameters are assertives for *monitoring* and the critical condition list represents *values* that support the *evaluation*. It turns out that the three control cycles and their subservices provide a very good framework to structure and integrate the VY management phenomena.

The BSRM model aims to provide a first graphical overview of the management services. For each management subservice identified, a more precise definition is to be made in a next step. This may be done using data models and data flow diagrams.

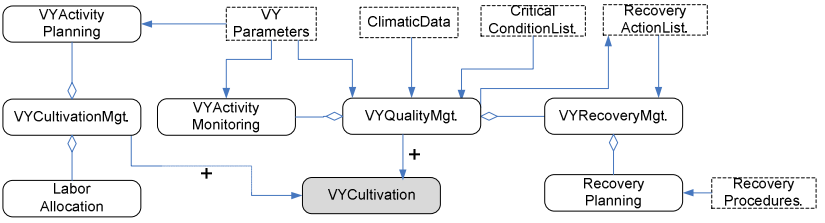

Fig. 7. BSRM model for vineyard cultivation service (management layer)

4.3 Robot Vacuum Cleaner

The next illustration is a fully automated vacuum cleaner [11]. The main goal of this vacuum cleaner is searching dust and keeping the room clean. To perform this task, it has sensors to detect dust, mop, and broom and dust box. The dustbin and the battery charging station are located in the building. Once the dust box is full the cleaner has to move to dustbin to empty it and as s the strength of the battery is low, it has to move to the battery station to recharge. Fig. 8a shows the core services of the vacuum cleaner.

Cleaning the room is the central service and it has two subservices namely *Move* and *CollectDust* – a service which collects dust by brushing and mopping. The cleaning room service involves at least two resources: room and the motor. This service converts the room into a cleaned room, so room is the service goal. The motor is defined as a resource to the main service (*CleaningRoom*), ensuring that it inherits to the subservices as well. There is another service related to the motor which is power supply (with battery). Finally the resources broom, mop and dust box are used by the *CollectDust* subservice.

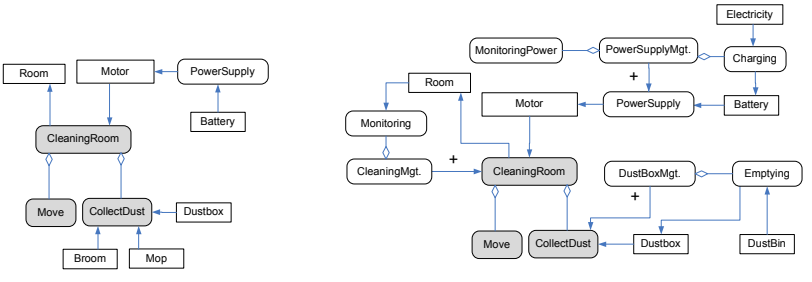

Fig. 8. (a) Core services (b) core + management services for automated vacuum cleaner

The next two steps are identifying management services. The service model for these steps is depicted in Fig. 8b. Since the robot is supposed to work autonomously, self-management is essential and only diagnostic control is relevant. It follows that the cleaning service has a cleaning management service which controls the cleaning process. Hence "monitoring" becomes a subservice of *CleaningMgt*. The next important management service is dust box management which intends to empty the dust box when it detects that dust box is full. *Emptying* works as a subservice.

PowerSupplyMgt monitors the supply of power to the battery and if it finds out that the battery strength is not on the required level, charging is executed.

A comparison of our approach with [11] reveals interesting information to the service designer. The modeling approach in [11] starts from an extended version of the Tropos [2] goal model. Then independently, it identifies non-intentional entities: external resources (room, dustbin and battery charger) and internal ones (dust box and battery). Entities are then related to the goals via the concept of *condition*, corresponding to the extended REA notion of policy constraint.

The management service modeling approach starts with the core services and identifies internal and some external resources needed or influenced by them. It then derives management services and management subservices. For example, *PowerSupplyMgt* has Monitoring and Charging subservices. The services have service policies. These include goals, but note that the goals are not defined globally, as in the previous approach, but per service. In the end, the entities identified are the same in both approaches, but the service-service and service-resource relationships in our model are not considered in the [11] approach.

5 Conclusion

In this paper, we have developed a design framework for management services, based on an established business ontology and applied to two case studies from the literature. The evaluation suggests that the framework provides useful and specific modeling support and that it is widely applicable. The theoretical relevance of the paper consists in the extended REA management ontology, as well as the three management control cycles that we have distinguished and described in SOA terms and that go beyond current work in adaptive systems considering a diagnostic control cycle only. The paper may also have practical relevance to service engineers interested in aligning business services and software services and to whom current service design methods do not give much specific support when it comes to service management design.

Although a design framework is not easy to evaluate in practice, we intend to strengthen the validation by applying it to real-world cases and extending the comparison with related work in management information systems, software engineering, multi-agent systems and the field of adaptive systems.

An interesting topic for future research is the design and development of generic management service software, e.g. to deploy a diagnostic service on the basis of a policy definition and a given environment (available services) only.

References

[1] Bartels, A.: Smart Computing Drives The New Era of IT Growth. Forrester (2009)
[2] Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J., Perini, A.: Tropos: An Agent-Oriented Software Development Methodology. Proc. AAMAS 8(3), 203–236 (2004)
[3] Cetina, C., Haugen, O., Zhang, X., Fleurey, F., Pelechano, V.: Strategies for Variability Transformation at Run-time. In: Proc. 13th Int. Software Product Lines Conf., SPLC (2009)
[4] Dobson, S., Sterritt, R., Nixon, P., Hinchey, M.: Fulfilling the Vision of Autonomic Computing. IEEE Computer 43, 35–41 (2010)

[5] Geerts, G., McCarthy, W.: Policy-Level Specifications in REA Enterprise Information Systems. Journal of Information Systems 20(2), 37–63 (2006)
[6] Hruby, P.: Model-Driven Design of Software Applications with Business Patterns. Springer, Heidelberg (2006)
[7] Kephart, J., Chess, D.: The Vision of Autonomic Computing. Computer, 41–50 (2003)
[8] Moscinat, A., Binder, W., Jazayeri, M.: Runtime Adaptability through Automated Model Evolution. In: Proc. 14th IEEE Enterprise Distributed Object Computing Conference, pp. 217–226 (2010)
[9] McCarthy W.E., The REA Accounting Model: A Generalized Framework for Accounting Systems in a Shared Data Environment. The Accounting Review, 544-577 (1982)
[10] Mueller, I., Han, J., Schneider, J.-G., Versteeg, S.: A Conceptual Framework for Unified and Comprehensive SOA Management. In: Feuerlicht, G., Lamersdorf, W. (eds.) ICSOC 2008. LNCS, vol. 5472, pp. 28–40. Springer, Heidelberg (2009)
[11] Morandini. M., Penserini. L., Perini. A.: Towards goal-oriented development of self-adaptive systems. In: Proc. Int. Workshop on Software Engineering for Adaptive and Self-Managing Systems, SEAMS (2008)
[12] Mason-Jones, R., Towill, D.R.: Information enrichment: designing the supply chain for competitive advantage. Supply Chain Management 2(4), 137–148 (1997)
[13] Papazoglou, M.P., van den Heuvel, W.J.: Service oriented architectures: approaches, technologies and research issues. VLDB Journal 16(3), 389–415 (2007)
[14] Peffers, K., Tuunanen, T., Rothenberger, M., Chatterjee, S.: A Design Science Research Methodology for Information Systems Research. Journal of Management Information Systems 24(3), 45–77 (2008)
[15] Porter, M.: Competitive Advantage. Free Press, New York (1985)
[16] La Rosa, M., Dumas, M., ter Hofstede, A.H.M., Mendling, J., Gottschalk, F.: Beyond Control-Flow: Extending Business Process Configuration to Roles and Objects. In: Li, Q., Spaccapietra, S., Yu, E., Olivé, A. (eds.) ER 2008. LNCS, vol. 5231, pp. 199–215. Springer, Heidelberg (2008)
[17] http://www.s-cube-network.eu/results/deliverables/wp-ia-2.2/CD-IA-2.2.2_Collection_of_industrial best_practices_scenarios_and_business_cases.pdf/view
[18] Searle, J.: A Classification of Illocutionary Acts. Language in Society 5, 1–24 (1976)
[19] Simons, R.: Performance Measurement and Control Systems for Implementing Strategy. Prentice Hall, Englewood Cliffs (2000)
[20] Salehie, M., Tahvildari, L.: Self-adaptive Software: Landscape and Research Challenges. ACM Transactions on Autonomous and Adaptive Systems 4(2), 1–42 (2009)
[21] UN/CEFACT Modelling Methodology (UMM) User Guide (2003), http://www.unece.org/cefact/umm/UMM_userguide_220606.pdf
[22] Vymetal, D., Hunka, F., Hucka, M., Kasik, J.: Enterprise modeling: process and REA value chain perspective (2010), http://mpra.ub.uni-muenchen.de/24617/
[23] Weigand, H., Johannesson, P., Andersson, B., Bergholtz, M.: Value-Based Service Modeling and Design: Toward a Unified View of Services. In: van Eck, P., Gordijn, J., Wieringa, R. (eds.) CAiSE 2009. LNCS, vol. 5565, pp. 410–424. Springer, Heidelberg (2009)
[24] Weigand, H., Johannesson, P., Andersson, B., Bergholtz, M., Jayasinghe Arachchige, J.: Closing the User-Centric Coordination Cycle. In: Proc. CAiSE 2010 Forum. LNBIP, vol. 72, pp. 267–282. Springer, Heidelberg (2010)
[25] Weigand, H., Heuvel, W.J., van den Hiel, M.: Business Policy Compliance in Service-Oriented Systems. Information Systems 36, 791–807 (2011)
[26] Yu, J., Sheng, Q.Z., Swee, J.K.Y.: Model-Driven Development of Adaptive Service-Based Systems with Aspects and Rules. In: Chen, L., Triantafillou, P., Suel, T. (eds.) WISE 2010. LNCS, vol. 6488, pp. 548–563. Springer, Heidelberg (2010)