Modelling Electronic Service Systems Using UML

James Skene[1], Giacomo Piccinelli[1], and Mary Stearns[2]

[1] Dept. of Computer Science, University College London, Gower Street
London WC1E 6BT, UK
{g.piccinelli, j.skene}@cs.ucl.ac.uk
[2] 2 HP Software & Solutions Operation, Pruneridge Avenue
Cupertino, CA 95014, USA
mary_stearns@hp.com

Abstract. This paper presents a profile for modelling systems of electronic services using UML. Electronic services encapsulate business services, an organisational unit focused on delivering benefit to a consumer, to enhance communication, coordination and information management. Our profile is based on a formal, workflow-oriented description of electronic services that is abstracted from particular implementation technologies. Resulting models provide the basis for a formal analysis to verify behavioural properties of services. The models can also relate services to management components, including workflow managers and Electronic Service Management Systems (ESMSs), a novel concept drawn from experience of HP Service Composer and DySCo (Dynamic Service Composer), providing the starting point for integration and implementation tasks. Their UML basis and platform-independent nature is consistent with a Model-Driven Architecture (MDA) development strategy, appropriate to the challenge of developing electronic service systems using heterogeneous technology, and incorporating legacy systems.

1 Introduction

The contribution of this paper is a UML profile for modelling systems of electronic services.

UML [16] is an object-oriented modelling language that has found broad application in analysis and design for software systems. A profile is a package of syntactic and semantic refinements for the language, which allows it to naturally model domains of interest.

An electronic service is a set of metadata, communication interfaces, software and hardware supporting a business service. A business service is a bundle of coordinated business capabilities (the content of the service) associated with provisioning mechanisms that establish the conditions under which clients, whether external or internal to the business, can access the capabilities of the service.

Business services encapsulated by electronic services benefit from additional communication and provisioning channels, but further, they permit the automated coordination of capabilities, resources and information, both within and

between organisations. This gives rise to Electronic Service Systems (ESSs), in which the services are integrated using auxiliary components such as workflow engines for coordination, databases to store knowledge about the state of the enterprise, and Electronic Service Management Systems (ESMS),which we characterise in this paper as combining these various capabilities to provide viewpoints and control of the enterprise to management, citing experience of the HP Service Composer and the DySCo (Dynamic Service Composer) research prototype.

The challenge on the business side is to adapt business infrastructure and models to service-oriented principles. The challenge on the technical side is to provide integration solutions that are accessible, comprehensive and beneficial. This requires thorough understanding and active management of the relationships between business capabilities and technical infrastructure. Such understanding and management can be achieved through modelling. This modelling serves as a starting point for software implementation, integration and provisioning tasks, which must be applicable to electronic services realised using a variety of technologies. The Model Driven Architecture (MDA) is a software development strategy based on UML models that explicitly addresses the challenge of integration of heterogeneous systems, and we therefore choose UML as a basis for our modelling, to ensure compatibility with this approach.

Determining strategies for coordinating services can be difficult, due to complex dependencies between services and the large number of possible states for the enterprise, arising from the parallel evolution of multiple services. We associate a formal model of behaviour based on workflows with our models of services, providing the opportunity for analysis. This model also formalises our notions of coordinated capabilities forming larger conceptual entities such as services, and the flows of information resulting from service enactment. Finally, the formal semantic provides a reference for implementation activities proceeding from our models, giving developers the opportunity to assert that software components act as required.

The remainder of this paper is structured as follows: In Section 2 we provide a background with a discussion of electronic services (Section 2.1), and UML and the MDA (Section 2.2); in Section 3 we present a meta-model describing the domain of ESSs; in Section 4 we show the translation of the meta-model into a UML profile; in Section 5 we discuss related work and then summarise in Section 6.

2 Background

2.1 Electronic services

The notion of a 'business service' enables the management within an enterprise of 'capabilities' to deliver some benefit to a consumer. The term 'capability' refers to the coordination of simpler tasks to achieve an end; the concept is used to raise the level of abstraction when describing the way that a business behaves. When describing business services, capabilities are divided into those involved

in 'provisioning' the service, and those providing the 'content' of the service. The content of a service is the set of capabilities that deliver the benefit of the service to the client. For example, the content of a freight service refers to the capability of moving goods from one place to the other. Provisioning refers to the business channel [6] between the provider and the consumer of a service. In the example, provision covers selection, product offer, pricing, and interaction processes that the freight company applies to its customers. Content and provisioning are complementary aspects of a service. On the one side, the provisioning logic depends on the capabilities that the provider can support. On the other side, the capabilities made available to consumers depend on the provisioning logic adopted by the provider. In the example, the option of delivery tracking might be made available only to selected customers. The example is based on previous research in the freight domain [12], and will be used throughout the paper.

An electronic service is a business service with communication and coordination aspects implemented using computer systems [14].

Because business services require communication between the provider and the consumer it is natural to provide interfaces to business services using communication technologies such as computer networks, and the software that supports this such as middleware for distributed systems. Indeed, a service metaphor is widely used in these technologies. Listeners on network interfaces are often referred to as services, and web-services communicate using Internet protocols to provide services of all sorts. Such services are closely analogous to business services, even to the extent of exhibiting a separation between provisioning capabilities in the form of meta-data interfaces, reflection and directories, and the back-end logic implementing the content of the service. Web-services conform to the model further, by including business terms in meta-data [24], enabling a market in services.

Despite the similarities, our notion of electronic service should not be confused with middleware services. Services must also be coordinated: Internally, to marshal the involved capabilities and resources and establish the relationship between content and provisioning; and externally, to manage the interaction between the service and its clients and environment. This coordination requires a view of the behaviour of a service. We therefore introduce an operational semantic for capabilities, presented in Section 3.2. This semantic is broadly compatible with workflow languages, suggesting that services could be both coordinated and enacted by workflow engines.

Our semantic also describes abstractly the effect that activities have on the information in their environment, for example the known locations of vehicles, or statistics such as the total revenue for a service. Such information can have a role in coordinating capabilities, and may be maintained and leveraged using databases or other accounting mechanisms.

There is also a need to manage the resources required by a service, which may be the role of an Enterprise Resource Planning (ERP) application. Generally, if electronic services are in place there will be the need and opportunity

to integrate them using a technical infrastructure. We introduce the notion of an Electronic Service Management System (ESMS), informally defined as an application that includes coordination, information and resource management capabilities, providing business-oriented viewpoints and control over the services that it manages.

IT technology trends and the service model for business provide the context for electronic services. An enterprise adopting an electronic service strategy would structure its business as services, provide interfaces to those services using middleware technologies, coordinate and automate the services from a workflow-oriented perspective and implement a technological infrastructure to take advantage of the coordination and communication opportunities that are the key benefit of the electronic service model.

2.2 UML and the model-driven architecture

Businesses adopting an electronic service strategy will be faced with integration and implementation challenges. Modelling systems of electronic services is a vital step towards meeting a number of these. When implementing a new electronic service, or adapting an existing business service to an electronic service it is necessary to understand the intended environment of the service, and its interaction with other services and management systems. Similarly, when introducing new management components, it is necessary to have a clear understanding of the services with which it will interact.

The Unified Modelling Language (UML) is an object-oriented graphical language that has found wide applicability in analysis and design for software engineering. In this paper we provide a profile for UML to allow the modelling of ESSs. Profiles are an extension mechanisms whereby the innate notations provided by the UML can be augmented with labels, called 'stereotypes', tagged values and constraints, which provide semantic refinement, annotations and syntactic refinement respectively.

UML is based on a conceptual architecture that is divided into four meta-modelling layers as shown in Figure 1. The lowest level is the data layer (M0), in which objects such as data-patterns in computer memory and other real-world phenomena including people and things are supposed to reside. The elements in the lowest level are classified by types in the UML models that analysts and designers produce, which hence reside at the next meta-level (M1). UML model elements are, in turn, objects of classes in the UML meta-model (M2). Attached to these meta-classes are semantic descriptions and syntactic constraints that control the meaning and applicability of the UML. The meta-model at level M2 is self-describing, so can also be regarded as residing in level M3 (and plausibly all higher levels).

Profiles then, are a means of refining classes, semantics and syntactic constraints at the M2 level. Confusingly, profiles exist at the M1 level, so that they can be denoted using UML and deployed by including them with any UML model that requires their language extensions. They can therefore be regarded as injecting 'virtual meta-classes' into the UML meta-model (M2).

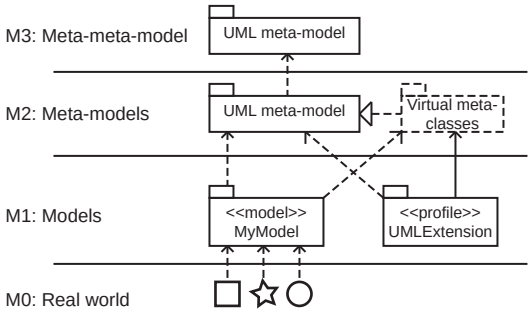

Fig. 1. Meta-modelling architecture of the UML

Before presenting our profile, we present a meta-model that is similar to the UML meta-model, and can be considered to reside at level M2 in the conceptual architecture. This is a common practice when defining profiles [5], as a new meta-model describes the semantic domain directly, independently of the need to refine the semantics of the UML meta-model. In section 4 the meta-model is mapped onto profile elements, and existing elements in the UML meta-model. Our meta-model therefore serves as both a reference model for our definition of electronic services and to define the semantics of the profile.

The Model Driven Architecture (MDA) [17] is a modelling approach based on UML. It recommends that development organisations separate models of their business logic (Platform Independent Models - PIMs), from technical artifacts, such as design models (Platform Specific Models - PSMs) and source code. The benefit is to insulate organisations from the cost of re-deploying software services as architectural infrastructures change, particularly middleware standards. The approach also supports the integration of heterogeneous and legacy software, and for these reasons is extremely well suited to development tasks in an electronic service environment. In the terminology of the MDA, the models produced using our profile are Platform Independent Models (PIMs). UML can represent refinement relationships between models, for example between a PIM and a PSM. Our models can therefore be related to more detailed design models, supporting the MDA approach when implementing or integrating electronic services.

In supporting an MDA approach our profile is similar and complementary to other profiles including the standard Enterprise Distributed Object Computing (EDOC) profile [18], which can be used to represent enterprise computing systems in a platform-neutral manner.

3 The ESS meta-model

The ESS meta-model is divided into two packages as shown in Figure 2. These partition the elements pertaining to services from those which represent manage-

ment applications. The management component metamodel naturally depends on concepts from the service metamodel. The following sections present these metamodels in detail.

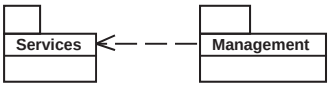

Fig. 2. Subpackages within the ESS meta-model

3.1 The service meta-model

Figure 3 shows the part of the services meta-model related to the composition of capabilities into services. The elements shown are now described:

Service An electronic service as described in Section 2.1. Services have any number of provisioning capabilities, and a single top-level content capability (the capability to deliver the service). Services can be composed of sub-services, in which case the content capability coordinates the content of each sub-service, and each sub-service must have a provisioning capability that makes a service offer to a role in the coordinating content capability.

Capability A business capability described by a workflow. The behaviour of capabilities is described formally in Section 3.2. Informally, a number of roles perform actions and cooperate to complete some task. Capabilities can be composed in a hierarchy. The workflow of a coordinating capability constrains the order of tasks in the component capabilities.

CapabilityRole A capability role identifies the behaviour of a worker or resource in a coordinated task. Capability roles can be assigned to actual business entities as discussed below.

InformationItem An identifier for a piece of information about an enterprise that is relevant to a task. Some workflow actions require information as a prerequisite and produce or process information as by-product of their enactment.

Observation Observations give rise to new information from existing information. This captures the idea that not all derived information is produced by a particular action. When the condition of the observation is satisfied then new information may be introduced by the observation expression.

Constraints defined over the meta-model further reinforce these informal semantics. For example, capabilities may not coordinate themselves. Constraints are expressed formally using OCL [16]:

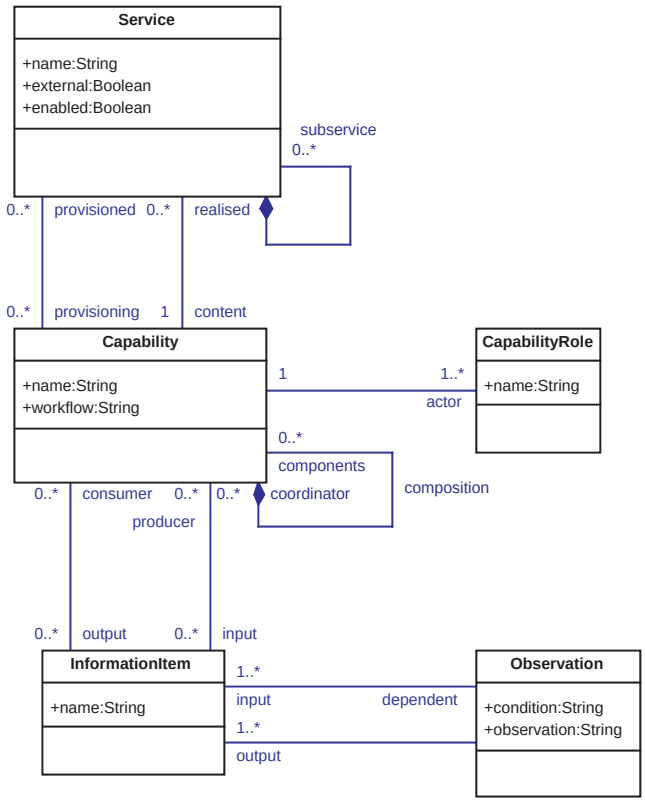

Fig. 3. Capabilities view of the services meta-model

context Capability
  def:
    let allCoordinators = self.coordinator→union(
      self.coordinator→collect(c | c.allCoordinators))
  inv:
    not self.allCoordinators→exists(c | c = self)

Complementary to the abstract view of services are models of the business assets in an enterprise, and their assignment to capability roles to realise a service. Figure 4 shows the meta-model classes supporting such models.

BusinessEntity A business entity is a person, resource or system that can fulfil one or more roles in a capability.

ServiceOffer A service offer is made to a capability role (typically that of the 'customer'). That capability role must be associated with one of the provisioning mechanisms of the service.

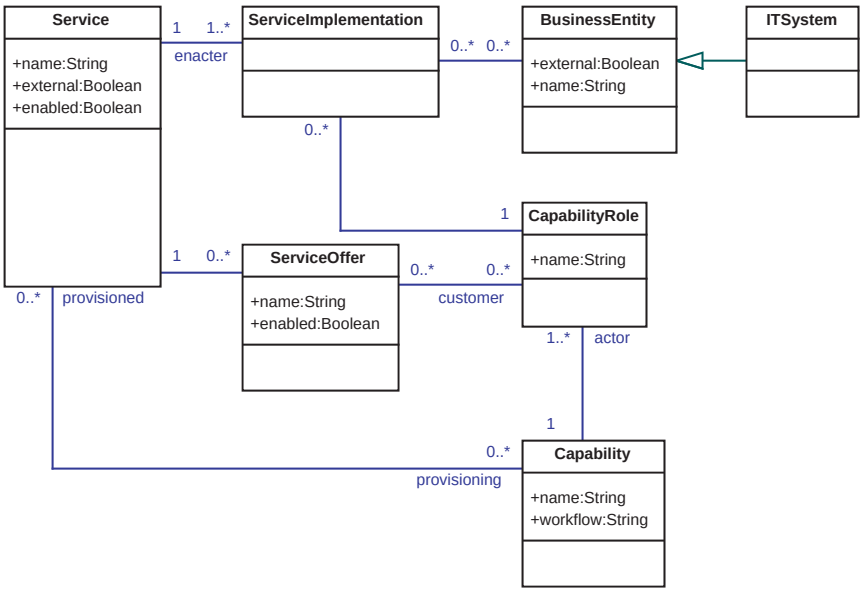

Fig. 4. Implementation view of the services meta-model

ServiceImplementation ServiceImplementation captures the idea that business assets can be assigned to capability roles in order to make a service concrete. There is no explicit notion of service instance. However, if necessary business assets can be grouped to show those relevant to a particular scenario.

ITSystem An IT system is a computing system that can perform a role in a capability. Electronic services are intended to provide integration and automated coordination. This class allows the identification of the components providing these services, possibly as a prelude to an MDA-style development activity. Section 3.3 provides refinements of this stereotype to identify likely management applications.

Additional classes not shown in Figures 3 and 4 are now discussed:

Property and HasProperties Properties capture different types of meta-data about capabilities. Such meta-information mainly refers to functional and non-functional requirements for a capability. For example, a property for a negotiation capability is to be usable only with a certain type of customers. The following classes inherit from HasProperties to enable the attachment of properties: BusinessEntity, CapabilityRole, Capability and Service. The properties mechanism maps onto the tagged-value mechanism in UML in the profile definition.

Group and Groupable Experience with the HP Service Composer revealed the benefit of composing capabilities into loosely-grouped higher-level ag-

gregates called 'clusters', in which capabilities exhibited functional overlaps, dependencies, mutual ownership or other subjective similarities. There is also often the need to group services into related offerings or 'service packs'. Finally, as stated above, a grouping mechanism addresses the lack of a concept of service instance by allowing the association of business entities that actually cooperate (since more than one entity can enact a given service role). Group and Groupable provide a single mechanism for hierarchical grouping. The following elements inherit from Groupable, and hence may appear in a Group: CapabilityRole, Capability, BusinessEntity, InformationItem, Service and Group. Grouping is implemented by UML's package mechanism in the profile definition.

3.2 Formal semantics for the service meta-model

In this section we formalise notions of information and coordination for capabilities, using the Structured Operational Semantics (SOS) style of [22], in which inference rules define the structure of a Labelled Transition System (LTS) intentionally. We do this independently of specific workflow languages, by omitting base cases for our rules. Instead, we assume that the workflow language allows us to make assertions such as:

$$\langle \sigma \cup I, C \rangle \xrightarrow{\alpha: I \to O} \langle \sigma \cup O, C' \rangle \tag{1}$$

Meaning that a specific, isolated capability, C, in a system where the current information is represented by $\sigma \cup I$, evolves to C' by undertaking an action, α, which effects some change, reflected by the transformation of the information I to new information O.

Capabilities may evolve independently of each other, when not coordinated:

$$\frac{\langle \sigma, C_i \rangle \xrightarrow{\alpha} \langle \sigma', C_i' \rangle}{\langle \sigma, \{C_1 \dots C_i \dots C_k\} \rangle \xrightarrow{\alpha} \langle \sigma', \{C_1 \dots C_i' \dots C_k\} \rangle} \tag{2}$$

Even when coordinated, capabilities may perform uncoordinated actions ($A(C)$ yields the set of actions that a process C can undertake):

$$\frac{\langle \sigma, C_i \rangle \xrightarrow{\alpha} \langle \sigma', C_i' \rangle \qquad \alpha \notin A(C_c)}{\langle \sigma, C_c[\{C_1 \dots C_i \dots C_k\}] \rangle \xrightarrow{\alpha} \langle \sigma', C_c[\{C_1 \dots C_i' \dots C_k\}] \rangle} \tag{3}$$

Coordinated actions may occur only when the coordinating process permits, and when all capabilities that can perform them are ready to do so simultaneously:

$$\frac{\langle \sigma, C_c \rangle \xrightarrow{\alpha} \langle \sigma', C_c' \rangle \quad \langle \sigma, C_1 \rangle \xrightarrow{\alpha} \langle \sigma', C_1' \rangle \dots \langle \sigma, C_i \rangle \xrightarrow{\alpha} \langle \sigma', C_i' \rangle \quad \alpha \notin \bigcup_{C_f \in F} A(C_f)}{\langle \sigma, C_c[\{C_1 \dots C_i\} \cup F] \rangle \xrightarrow{\alpha} \langle \sigma', C_c'[\{C_1' \dots C_i'\} \cup F] \rangle} \tag{4}$$

A capability may have multiple coordinators in the metamodel. The interpretation of this is that the capability is a subcapability of its coordinator. It is therefore replicated for each coordinator. Shared sub-capabilities are not synchronized.

Note that an action may require certain information to be exist and take a particular value before the action can be performed. Hence, coordination by shared memory is also possible for capabilities. Under the electronic service model, provisioning and content capabilities are not explicitly coordinated, hence this mechanism links these capabilities for a service. The provisioning capabilities create conditions under which the content capabilities are enabled.

Information in the system may arise naturally from the occurrence of actions. However, the progress of the system may depend on broader observations than those made in the context of a particular action. Hence we enable the modelling of observations that derive new information from that already present in the system:

$$\frac{\langle \sigma \cup I, \Gamma \rangle \wedge \exists o : I \rightarrow O \in \Omega}{\langle \sigma \cup I \cup O, \Gamma \rangle} \tag{5}$$

We do not prescribe the language used to specify observations. OCL would be a good candidate. A boolean expression could determine when the observation applied, and then **let**-clauses could introduce new information. Note that it is possible to specify observations that lead to inconsistencies in the system information. Modellers should try to avoid this. One strategy for dealing with this is to rule that if multiple values can be derived for an information item then that information is not known. However, in systems where action is preferable to inaction, this may not be safe.

For the purposes of assigning work the underlying workflow language must also associate actions with roles, although this association is not required in this discussion of coordination, as we assume that coordination is independent of the entities that implement roles. That is, an entity will eventually be capable of enacting all actions required of it during the evolution of the system.

The benefit of a formal semantic based on an LTS are in terms of simulation and analysis. A tool such as LTSA [13] can provide scenario-based validation of models. This can be used to assert safety conditions, fairness and liveness conditions, and to ensure the absence of deadlocks (presumably arising from capabilities failing to establish adequate preconditions for their successors). The use of information for coordination complicates such models, and can increase their state-space beyond feasibility. However, reasonable abstractions can usually be found.

3.3 The management meta-model

The management meta-model shown in Figure 5 allows the identification of common management components and their relationship to electronic services. We have not included modelling functional or structural relationships between

management components as this is out of scope of our discussion of electronic services. However, such modelling is necessary and is supported by the full expressive power of the UML, possibly augmented by other profiles such as the EDOC profile.

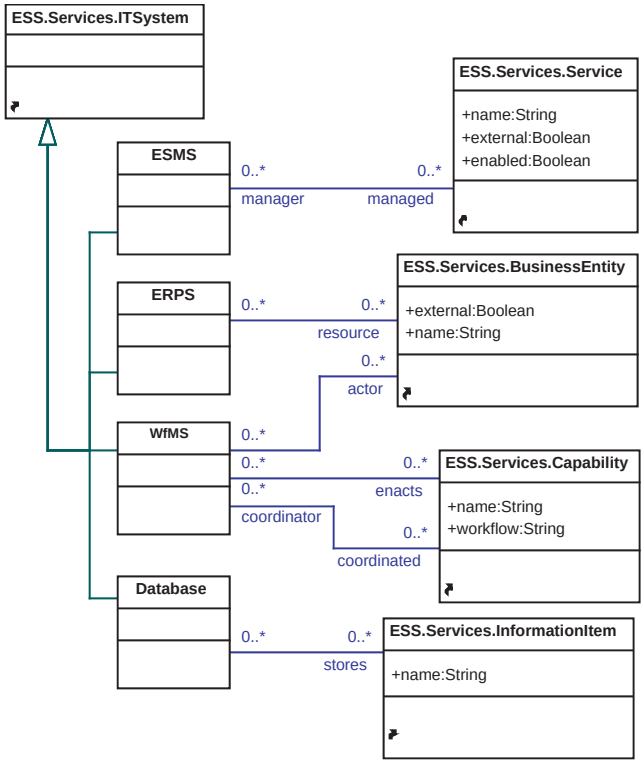

Fig. 5. The management meta-model

ESMS An application offering a enterprise-oriented management view of an electronic service environment. For example, the HP Service Composer [7], or the DySCo research prototypes [21]. Other candidate technologies might be an application service offering a middle-tier of business logic, with a web-server providing the management interfaces.

WfMS A workflow management system, either embodying a capability (enactment) or coordinating a number of subcapabilities. Examples of workflow applications are IBM's MQ-Series Workflow [8] and PeopleSoft's [20] PeopleTools and Integration Broker.

ERPS An Enterprise Resource Planning System, dedicating to coordinating entities in the system, presumably making them available to fulfil capabil-

ity roles. We do not consider resource planning in this paper, although it interacts at a functional level with coordination based on capabilities, and future work may provide a combined modelling approach. Examples of ERP systems are SAP's mySAP [23] and Baan's iBaan [1].

Database Most enterprises use databases to store information about the enterprise. Establishing a relationship between the (conceptual) information items and the databases that store them allows a modeller to check whether the information required by a business entity to fulfil a capability role is available in its context. Popular databases are Oracle [19] and MySQL [15].

4 The ESS Profile

The following tables relate elements in the meta-model to profile elements and elements in the UML meta-model.

Meta-model element	**Stereotype**	**UML base class**	**Parent**	**Tags**
Service	Service	Class	–	external enabled
Service.content	content	AssociationEnd	–	–
Service.provisioning	provisioning	AssociationEnd	–	–
Service.component	component	AssociationEnd	–	–
Capability	Capability	Class	–	
Capability.input	input	AssociationEnd	–	–
Capability.output	output	AssociationEnd	–	–
CapabilityRole	CapabilityRole	Class	–	–
InformationItem	InformationItem	Class	–	–
Observation	Observation	Class	–	condition observation
Observation.input	input	AssociationEnd	–	–
Observation.output	output	AssociationEnd	–	–
BusinessEntity	BusinessEntity	Class	–	–
ServiceOffer	ServiceOffer	Class	–	enabled
ServiceImplementation	Fulfills	Association	–	service
ITSystem	ITSystem	Class	BusinessEntity	–
ESMS	ESMS	Class	ITSystem	–
WfMS	WfMS	Class	ITSystem	–
WfMS.actor	wfactor	AssociationEnd	–	–
WfMS.enacts	enacts	Class	–	–
WfMS.coordinated	coordinates	Class	–	–
ERPS	ERPS	Class	ITSystem	–
Database	Database	Class	ITSystem	–

Table 1. Stereotypes in the ESS profile

Meta-model element	Tag	Stereotype	Type	Multiplicity
Service.external	external	Service	Boolean	0..1
Service.enabled	enabled	Service	Boolean	0..1
Capability.workflow	workflow	Capability	String	0..1
Observation.condition	condition	Observation	String	1
Observation.observation	observation	Observation	String	1
ServiceImplementation.service	service	Fulfills	Class	1
BusinessEntity.external	external	BusinessEntity	Boolean	0..1
ServiceOffer.enabled	enabled	ServiceOffefr	Boolean	0..1

Table 2. Tags in the ESS profile

All name attributes in the meta-models map to the name attribute of the class element in the UML meta-model. All associations in the meta-model map to associations in models. Stereotypes on AssociationEnds are used to disambiguate associations where more than one exists between the same two meta-model elements. The meta-model constraints also have translations into constraints on the profile elements, and additional constraints reflect the structure of the original meta-model. For example, the 'Fulfills' stereotype can only be attached to an association between a CapabilityRole and a BusinessEntity, and its service tag must always be present:

package Foundation::Core
  context Association
    inv:
      self.stereotype→exists("Fulfills") **implies**
        self.connection.participant.stereotype→exists("CapabilityRole")
        and
        self.connection.participant.stereotype→exists("BusinessEntity")
        and
        self.taggedValue.type→exists(name = "service")

5 Related work

The definition and characteristics of the ESS model derive substantially from the experience of HP Service Composer. UML notation is used in the HPSC, with a separation between platform-dependent and platform-independent models of an electronic service. Workflow notation and technology is used to model and manage the business logic of a service.

The ESS model is also closely related to the DySCo (Dynamic Service Composer) [21] research prototype. DySCo is the result of a two-year project involving University College London (UK), the University of St. Petersburg (Russia), the University of Ferrara (Italy), the University of Hamburg (Germany), and Hewlett-Packard (UK and USA). The objective of DySCo was the development

of a conceptual and technology framework for the dynamic composition of electronic services. While lacking direct support for UML, DySCo provides modelling facilities for workflows and a homogeneous execution platform for an ESMS.

An electronic-services model is currently being used in the context of the EGSO (European Grid for Solar Observations) [2] project. The model-driven approach to the architecture of the service provision part of the EGSO grid is expected to address the need to integrate services based on different provision models and execution platforms. Each service provider in the EGSO grid will be equipped with an ESMS. In addition, a specific ESMS federates and manages the service provisioning capabilities of the overall EGSO grid.

The Enterprise Collaboration Architecture (ECA) defined in the OMGs EDOC specification [18] provides a comprehensive framework for the modelling of enterprise systems. The ESS profile introduces enterprise system components that can be designed based on the ECA, and provides a means to model features peculiar to electronic services that are not explicitly addressed by the ECA. Similar considerations apply for the Reference Model for Open Distributed Processing (RM-ODP) [9], which is also closely related with the ECA.

Most technology and conceptual frameworks for electronic services [11] focus on web-service-based automation of the front-end of individual services. Web Services [3, 4] constitute the reference model for access to and basic orchestration of business resources. We envision Web Services playing a fundamental role in the realisation of electronic services. Still, a more comprehensive approach is needed for the realisation and operation of business-level services. An example of the issues involved in the realisation of business-level service is HiServs Business Port [10]. FRESCO (Foundational Research on Service Composition) [22] provides an example of second-generation framework for electronic service management. The focus of FRESCO is on the provision aspects of services.

6 Conclusions

Electronic services provide the conceptual and technology framework for the aggregation and coordination of business resources. The realisation and operation of a service requires close integration between different systems. A model-driven approach to development in an electronic-service environment helps tackle the integration issues arising from heterogeneity and change.

In this paper, we present a means to model ESSs using UML, in a manner compatible with the MDA approach. We apply concepts derived from the specific experience of HP Service Composer, but also closely related to concepts in OMGs EDOC specification and the RM-ODP. The semantics of the models are described with reference to a meta-model from which a UML profile is defined. The behaviour of electronic services is described formally using operational semantics, providing an additional benefit of our models as a foundation for formal analyses.

References

1. Baan. *iBaan*. http://www.baan.com/.
2. R. D. Bentley. EGSO – the european grid of solar observations. In *European Solar Physics Meeting, ESA Publication SP-506*, 2002.
3. E. Cerami. *Web Services Essentials*. ORielly and Associates, 2002.
4. M. Clark et Al. *Web Services Business Strategies and Architectures*. Expert Press, 2002.
5. D. S. Frankel. *Model Driven Architecture: Applying MDA to Enterprise Computing*. John Wiley and Sons, 2003.
6. B. Gibb and S. Damodaran. *ebXML: Concepts and Application*. John Wiley and Sons, 2002.
7. Hewlett-Packard Company. *HP Service Composer User Guide*, 2002.
8. IBM. *Websphere MQ Workflow*. http://www-3.ibm.com/ software/ integration/ wmqwf/.
9. ISO/IEC, ITU-T. *Open Distributed Processing – Reference Model – Part 2: Foundations, ISO/IEC 10746-2, ITU-T Recommendation X.902*.
10. R. Klueber and N. Kaltenmorgen. eServices to integrate ebusiness with ERP systems – the case of HiServs business port. In *Workshop on Infrastructures for Dynamic Business-to-Business Service Outsourcing (CAISE-ISDO)*, 2000.
11. H. Kuno. Surveying the e-services technical landscape. In *Workshop on Advance Issues of E-Commerce and Web-Based Information Systems (WECWIS)*. IEEE Press, 2000.
12. N. Linketscher and M. Child. Trust issues and user reactions to e-services and e-marketplaces: A customer survey. In *DEXA Workshop on e-Negotiation*, 2001.
13. J. Magee and J. Kramer. *Concurrency: State Models and Java Programs*. John Wiley and Sons, 1999.
14. A. Marton, G. Piccinelli, and C. Turfin. Service provision and composition in virtual business communities. In *IEEE-IRDS Workshop on Electronic Commerce*, Lausanne, Switzerland, 1999.
15. MySQL AB. *MySQL Database*. http://www.mysql.com/.
16. OMG document formal/2003-03-01. *Unified Modelling Language (UML), version 1.5*, January 2003.
17. OMG Document ormsc/01-07-01. *Model Driven Architecture (MDA)*, July 2001.
18. OMG, document ptc/02-02-05. *UML Profile for Enterprise Distributed Object Computing Specification*, May 2002.
19. Oracle. *Oracle database products*. http://www.oracle.com.
20. PeopleSoft. *PeopleTools and Integration Broker*. http://www.peoplesoft.com/.
21. G. Piccinelli and L. Mokrushin. Dynamic e-service composition in DySCo. In *Workshop on Distributed Dynamic Multiservice Architecture, IEEE ICDCS-21*, Phoenix, Arizona, USA, 2001.
22. G. Piccinelli, C. Zirpins, and W. Lamersdorf. The FRESCO framework: An overview. In *Symposium on Applications and the Internet (SAINT), IEEE-IPSJ*, 2003.
23. SAP. *mySAP*. http://www.sap.com/.
24. UDDI.org. *UDDI (Universal Description, Descovery and Integration) Executive White Paper*, November 2003. http://www.uddi.org/ pubs/ UDDI_Executive_White_Paper.pdf.