Towards a Model-Based and Incremental Development Process for Service-Based Systems*

Martin Deubler, Johannes Grünbauer,
Gerhard Popp & Guido Wimmel
Institut für Informatik
Technische Universität München
Boltzmannstr. 3, D-85748 Garching
{deubler, gruenbau, popp, wimmel}@in.tum.de

Christian Salzmann
BMW Car IT
Petuelring 116
D-80809 München
christian.salzmann@bmw-carit.de

ABSTRACT
In this paper we introduce the concept of service-based modeling and integrate the modeling techniques into existing development processes. We focus on services in the different phases of the modeling and introduce appropriate models as well as suitable notations. A small example from the automotive domain is used throughout the paper to illustrate the approach.

KEY WORDS
Service-Based Systems, Software Design and Development, Software Methodologies, Software Architecture, Automotive Software, Model-Based Development

1 Introduction

Software is getting a leading role in automotive development and innovation. It is stated that software and electronics account for 90 percent of all innovations in modern cars. Already today cost for software and electronics in current premium cars make up 40 percent of the overall cost. The next generation of premium cars is expected to host a cumulated amount of up to one gigabyte of binary code of software deployed via a set of embedded platforms. The types of applications in the automotive domain are not anymore limited to classical embedded systems, such as airbag control software, but cover a broad range from mission critical embedded systems in the X-by-wire field to infotainment and personalization in the MMI (Man Machine Interface) area. The MMI is a central software system in the car that manages the access of the user (driver) to the functions of the car such as infotainment, phone and Internet access, climate control and navigation system. The actual functions are not deployed on the MMI platform itself, but are distributed over a set of ECUs (Electronic Control Units) that are interconnected via different bus systems. In the actual 7 series of BMW, for example, the MMI system consists of a set of about 270 functions that can be triggered by the user and are deployed over about 40 ECUs.

A fundamental difference of such a system in comparison to classical applications is in the interaction and dependencies of the functions. One function, such as the volume control of the amplifier, can be triggered not via one single user interface, but by a set of other functions, e.g. when the navigation system informs the driver with a voice output the software lowers the volume of the audio system for the time of the voice information.

Since we have to deal with multiple function interactions, strong interrelations and dependencies between functions, we call this type of system *multifunctional system*, where one function may be used in different contexts. In such a multifunctional system, the functional aspect is more important than the whole application. For this reason we have to deal, apart from the functions themselves, with the encapsulation of them, their dependencies and their combination with other functions. The unit of function with its encapsulation, dependencies and combination is called a *service*.

Today's development methods are purely function oriented or in the newer forms of object oriented development methods they are use case oriented. Such function and object oriented methods are a priori not suited for such a flexible modeling of services, but they are open to adapt new development strategies.

In this paper, we suggest such an adaptation of existing development methods to service-based modeling, i.e. we integrate the service representation into a development process. In this context we also look at suitable notations for the representation of services in different diagram types and we identify necessary and appropriate models for a service-oriented development process.

2 Description of the Development Process

In this section we present a development process for service-based systems. The basis for our development process is a phase oriented development process like the waterfall model [2] or object oriented development methods such as the Unified Software Development Process [3] or the Catalysis Approach [4], just to name a few of them. We integrate services as a central concept and guideline of the

*This work was supported by the Bavarian high-tech funding program (High-Tech Offensive), within the project MEWADIS [1].

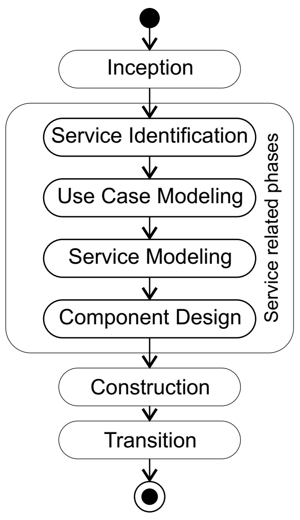

Figure 1. Phases of the Service Oriented Development Process

development. The presented process is model-based, i.e. the development is driven by explicit models of both the development artifacts (product model) and of the process itself. The models of the development artifacts have a predefined structure (which does not rule out textual parts), and the process model describes how development proceeds in the different phases, in terms of the product model. The process is incremental in that it can be repeatedly applied to add new functionality in small steps, which considerably reduces risk (see Section 2.5).

Figure 1 shows the phases of the service oriented process. We give a short overview about them and explain the service related ones in more detail in the following sections.

We assume that we have an *Inception Phase*, in which the project is born and both a project mission and the requirements are elaborated. This is the starting point of our process. The concept of service does not yet appear and thus the Inception Phase is not affected by our approach. The results of this phase are documented in a project mission document and in a requirements specification list.

After the Inception Phase the sequences of actions at a high abstraction level are modeled in the *Service Identification Phase*. In this phase a separation of the system takes place. The service identification is covered in Section 2.1. Results of this phase are activity diagrams modeling the run of service functions.

A first elaboration of the services, which are identified in the Service Identification Phase, is worked out in the *Use Case Modeling Phase*. The flow of events of every service function (as a subfunction of a service) is specified as well as the input and output data, preconditions for the processing, security requirements and service dependencies. The Use Case Modeling Phase is discussed in detail in Section 2.2. As a result this phase leads to a use case model with a structured textual specification of the use cases, sequence diagrams for one or more flows of events for a service function, a textual description of the security requirements, and a *logical service architecture*.

The sequence diagrams from the Use Case Modeling Phase make up a first version of the analysis model, which will be worked out in the *Service Modeling Phase*. The behavior of a service is specified formally in an abstract way by relating its inputs and outputs, e.g. using state transition diagrams. Execution scenarios are derived as compositions of services (logical architecture), and the security requirements are concretized in terms of the formal model. The service modeling is described in detail in Section 2.3.

The next phase, the *Component Design Phase*, ends the service specific activities in the process. Here the services are mapped to system components and thereby the logical architecture is transformed into a system architecture. This phase is summarized in Section 2.4. Result of this phase is a set of components with assigned services and a behavioral modeling of the components.

Afterwards, the system development is commonly (e.g. according to [5]) continued with the *Construction Phase* and the *Transition Phase*. These phases can be carried out conventionally, as service-specific issues have been resolved previously by the above mentioned mapping.

We explain our approach by referring to an industrial case study from the automotive domain, an onboard diagnostic system. The purpose of this system is to monitor devices of an automobile for malfunctions, to conduct diagnoses by downloading device-specific diagnosis software into the MMI system, and to suggest and take appropriate measures such as initiating a software update or offering to contact a dealer's workshop with relevant information for them to prepare a repair. As an example for a monitored device, we consider a climate control unit.

2.1 Service Identification

In this phase, requirements have to be divided and they have to be arranged to actors, whereby actors can be represented by roles, systems or services.

In a first step the requirements of the requirements specification list from the former phase are transformed to flows of activities, which are sufficiently fine-grained such that each activity can be carried out by one actor.

In a second step after this division we have to arrange these activities to their executing actors, i.e. the actor who gets information from this activity or who sends information to it. In the model the actors are swim lanes within an activity diagram, the nodes are the activities and the arrows show their causal (and temporal) relationship.

Since we deal with service-based modeling we have to extend the actor model. Conventionally we have to deal with two types of actors. The first actor type is the abstraction of a real person within a role, e.g. a driver role as abstraction for a person who drives and operates a car. The second actor type represents external systems interacting with the system to be developed (e.g. via the different automotive bus systems) which are not modeled in our specification. For service-based development, we add a third actor type, a *service*. The system to be developed is not modeled by one actor but rather by a set of actors of type service, i.e. small self-contained functional entities responsible for a number of activities belonging together. Since services interact both with the above mentioned two actor types and with other services, we can treat service-interactions in the same way as other actor-interactions.

In the context of service modeling, we call the activities performed by actors of type service *service functions*; otherwise we call them *actions* (human actor) and *system functions* (other interacting system), respectively. The needed functionality of a service is given by all service functions assigned to the corresponding service actor. In such a way we build up a usability driven model of services and service functions.

For a better understanding of the diagrams we have annotated the swim lanes with the belonging actor type symbol, whereby the stickman (Figure 2a) represents a human actor and the box symbol (Figure 2b) stands for an interacting system. For our extended actor model we introduce a new symbol for the actor type *Service*, which is shown in Figure 2c.

Example An example of an activity flow from our case study is given in Figure 3, namely for the requirement *Onboard Diagnosis*. We have to deal e.g. with the human actor *Driver*, the subsystem actor *Automotive* and the service actors *Display*, *Climate Control Unit* and *Onboard Diagnostic* all represented by swim lanes. Service functions are e.g. *show error message* or *start onboard diagnosis*.

2.2 Use Case Modeling

In the last years, the concept of *Use Cases* [6, 3, 7, 8] has become widely accepted within object oriented development methods. The basic idea of this approach is that both the domain objects as well as the user interaction of the system are modeled within early development phases. The use cases are more than a construct for capturing system requirements: they drive the whole development process and they provide major input when finding and specifying classes, subsystems, interfaces and test cases (for more information cf. [3]). Furthermore, use cases are adequate for an iterative development, as we will see in Section 2.5.

While uses cases have already been integrated into object oriented development, they do not cover aspects of

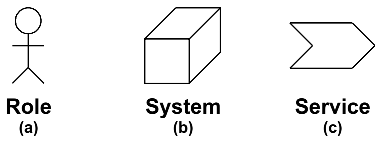

Figure 2. The Extended Actor Model

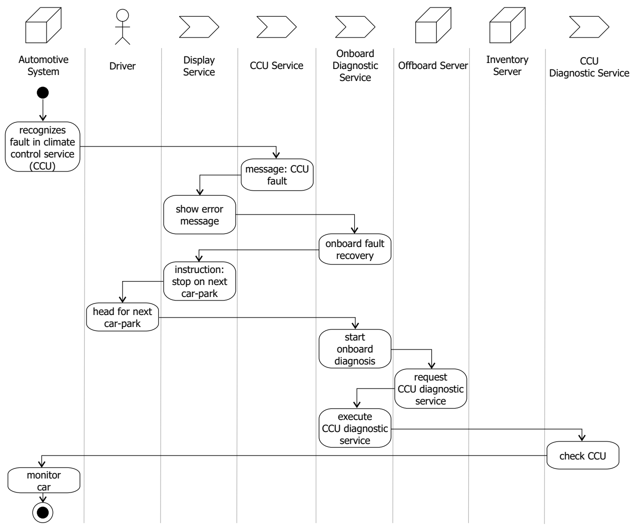

Figure 3. Activity Diagram for the Requirement “Onboard Diagnosis”

service-based modeling. In this case, we have to deal with the following particularities:

There is no need for a *common domain model*, which has to be worked out e.g. in information systems during the business modeling and has to be refined during the Use Case Modeling Phase. Services only deal with a subset of system objects, the ones we need for information storage within the services and as input and output objects for the service.

Furthermore, we consider *security requirements* within this phase. In common systems, we describe use cases in a structured textual way. Beside the actors a textual description covers the processing, variants illustrate peculiarities in the processing, the types of input and output data are specified and often a precondition for the execution is given. Here we add a security section where we describe possible threat scenarios for every security objective. Thereby we have to check possible violations of the objectives confidentiality, authenticity, integrity, non-repudiation and availability. More information about security within the requirements phase is given in [9].

Finally we add a section “involved services”, that highlights all concerned services. In a later step we can refine the use case and connect the involved services by relations like *affects*, *uses* or *controls*, which leads to a *logical service architecture*. It provides us a structured view on the system functions and how they are related (cf. [10, 11]), and eventually a chance to cope with feature interaction problems from the beginning of the development.

Thus we have a structured textual description. For a complete service use case model we have to include every service function we have identified in the Service Identification Phase (cf. Section 2.1) in such an extended use case. Note that this is a partial description, since the activity flow diagram shows just an exemplarily run of the service. We

Table 1. Use Case Description for the Partial Onboard Diagnostic Service

Field	**Description**
Use Case	On Board Diagnostic
Actors	Driver, Automotive System, Offboard Server, CCU Service, Onboard Diagnostic Service, Display Service, CCU Diagnostic Service
Precondition	Engine running, hand brake tightened
Processing	1. Onboard Diagnostic Service checks error cause 2. Inform Driver 3. Download CCU Diagnostic Service from Offboard Server 4. Start CCU Diagnostic Service
Variants	(a) Error cause can't be detected (b) Error can be fixed by CCU Diagnostic Service, no download required (c) No download possible (various reasons) (d) Termination by Driver, resumption at a later date
Input	Message on CCU fault, data from system test, confirmation message by Driver after notification, downloaded CCU Diagnostic Service
Output	Status Message for Driver, diagnosis request, download request, start signal for CCU Diagnostic Service
Security Objectives	Confidentiality: *no threats*; Authenticity: *no threats*; Integrity: *to ensure for internal communication*; Non-Repudiation: *no threats*; Availability: *possibly Internet connection, non-critical*
Involved Services	Display Service, CCU Service, CCU Diagnostic Service

obtain a complete service description by merging all partial use case descriptions.

In addition to the textual description we build in the Use Case Modeling Phase first analysis diagrams in form of sequence diagrams. For each service function with its belonging textual description we model the textually specified processing within a sequence diagram, where we depict the message flow between the different actors. In this diagram our introduced symbol for services (cf. Figure 2) comes into action, where we use it to mark services as special actor types.

In the following we sum up the steps which have to be done for service functions to model them as use cases. Note that we have no step for modeling an object model, because we do not have a domain model in service-based modeling as mentioned above.

1. Elaborate structured use case descriptions.
2. Specify threats and security objectives and add them to the textual use case descriptions.
3. Identify service relations and add them to the textual use case descriptions.
4. Formalize the use case descriptions exemplarily in one or more sequence diagrams.

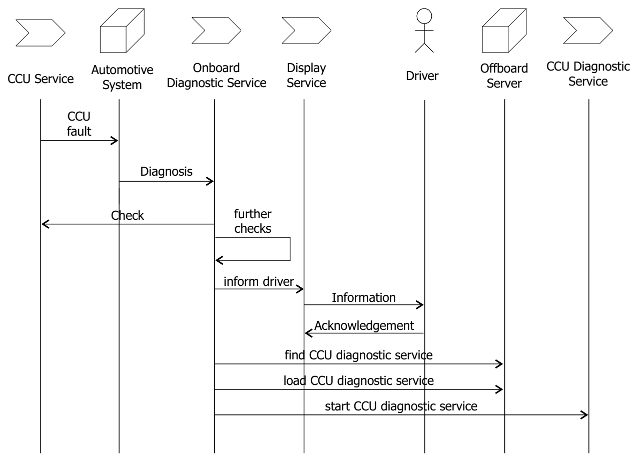

Figure 4. Sequence Diagram for Use Case

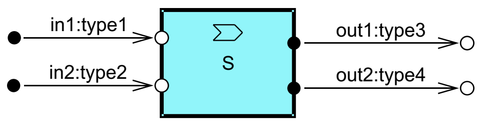

Figure 5. Service Actor

Example Table 1 shows the use case description for the Onboard Diagnostic Service according to Figure 3. Figure 4 shows the corresponding sequence diagram for our use case.

2.3 Service Modeling

In the Service Modeling Phase, we use the sequence diagrams and the textual use case descriptions developed in the Use Case Modeling Phase as a basis to develop a more detailed analysis model. The analysis model we employ is a formally (i.e., mathematically precisely) defined model for service architectures; it is described in more detail e.g. in [12]. We make use of tool support for these models in form of the CASE tool AUTOFOCUS [13], which offers graphical description techniques similar to a subset of UML-RT.

The analysis model of a service-based system consists of a number of actors, which are of the three actor types described above. For the development of service-based systems, we focus on the specifications of the service actors. Human roles and external systems form their environment.

Figure 5 shows the representation of a service S. Services have an interface consisting of input and output ports (denoted by empty and filled circles) and communicate with their environment via typed input and output channels connected to the ports. The arrow symbol (the same as we used in the use case modeling, cf. Figure 2c) at the top of the box identifies this actor as a service.

We specify the behavior of services as partial functions from sequences of inputs to sets of sequences of outputs. Partiality is characteristic for services in that only the

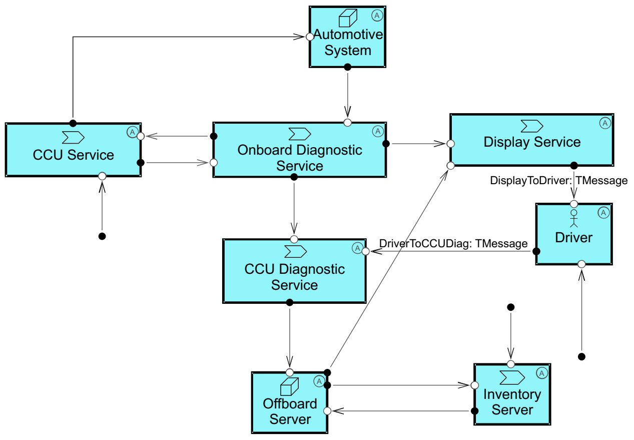

Figure 6. SSD of the “Onboard Diagnostic”

service relevant behavior is specified. The security requirements are concretized in terms of this model.

In the CASE tool AUTOFOCUS, the behavioral functions are specified in an executable way, as state transition diagrams. AUTOFOCUS can be used to model an actual execution scenario, given as a network of actors connected via the channels. These models provide a basis for consistency checks, behavioral verification of safety and security properties specified in temporal logic, test case generation, generation of monitor code, and further development (Component Design Phase).

Example In Figure 6, we find the system structure diagram (SSD) of the execution scenario corresponding to the sequence diagram in Figure 4. We added safety and security properties to the model and verified them using the SMV model checker [14] connected to AUTOFOCUS. At the modeled abstraction level, model checking performance was not an issue as the computation time is in the order of a few seconds per property. For example, one of the required properties for the onboard diagnosis system is that before the CCU diagnostic service can be started, the driver must have been instructed (via the display service) to drive to a parking lot and halt his car. In our formalism, this is specified by “precedes(DisplayToDriver == DriveToParkingLot, DriverToCCUDiag == startDiag)”, which could be shown to be true for the model.

2.4 Component Design

The logical service architecture provides a functional view on the system, i.e. the definition of services and service dependencies, respectively. Until the Component Design Phase there are no structural constraints taken into account. Here, the services can be mapped to one or even a number of component architectures.

However, designing components with respect to composing services into components is not the focus of this paper. For detailed investigations see for instance [15] or [12].

Example In our example, the service infrastructure is an OSGi [16] platform, a Java framework supporting the deployment of services. The services are mapped to a number of so-called OSGi “bundles”: an “Onboard Diagnosis”, a “Software Download” and an “CCU Diagnosis” bundle.

2.5 Incremental Development

In the waterfall model, the system is analyzed as a whole, designed as a whole, implemented as a whole and tested as a whole. Feature interactions are often recognized in later development phases and it is very difficult, expensive and time-consuming to solve such kinds of problems. For this reason, especially object oriented systems are built up over time incrementally rather then all at once near the end when change become expensive [3]. Another reason for applying an incremental development process is that the staff can work more effectively.

In the domain of service-based modeling we begin an iteration with the elaboration of one or more requirements from the requirements specification list into an activity flow for each requirement. This elaboration is done in the Service Identification Phase, as described in Section 2.1.

Afterwards, in the Use Case Modeling Phase and in the Service Modeling Phase (see Sections 2.2 and 2.3) the service functions, which are identified and specified in the logical service architecture, are modeled in detail.

Services, which consist of sets of service functions, can now be mapped to system architectures in the component design. The construction and transition phases can be carried out, because all service functions corresponding to the currently elaborated requirements have been modeled so far.

In a next increment new requirements are transformed into activity flows. It is possible that either new service functions will be added to a formerly modeled and constructed service (i.e. the service will be enriched with additional functionality), or new services need to be elaborated. The new service functions have to be modeled in the Use Case Modeling Phase and in the Service Modeling Phase and afterwards to be added in the component design and in the construction as well as in the transition. The logical service architecture evolves due to changed or newly introduced service relations, which results both from the inclusion of new service functions and from the elimination of early discovered (undesirable) feature interactions, of increased service complexity or contradictions.

We repeat the iterations with further requirements from the requirements specification until all requirements are treated.

Example In a second iteration in the presented case study, we intend to add a feature to the diagnostic services to carry out software updates for the corresponding devices.

3 Conclusion and Future Work

To cope with the increasing complexity and the large number of mutually dependent functionalities in current and future systems (as in the automotive domain), we presented a model-based development approach based on the concept of services as the basic building blocks in the elaboration phase of development. We abstract as long as possible from concrete component architectures but instead focus on the system as a logical network of units of function (the services) with abstract behavioral patterns and relationships to other services.

We sketched a phase oriented process related to the Unified Process, defined in terms of particular modeling techniques to be used in the different phases. We integrated the concept of services into these phases. Services are introduced in the service identification phase by defining a new actor type and refined during the following phases until they are mapped to components. An important feature of our approach is the consideration of safety and security aspects throughout the development. The behavioral service models are defined in formal terms and supported by the CASE tool AUTOFOCUS, allowing for verification, simulation or test case generation for service-based scenarios. Our process can be applied in an iterative and incremental way. We demonstrated the process at the example of an industrial case study carried out in cooperation with a major car manufacturer.

In the area of telecommunications the concept of service (there called feature) has been used for quite some time. More recently, frameworks have been developed to offer Web services in Internet-based architectures. Only lately, services have been applied in the domain of embedded systems such as in automobiles. These approaches primarily focus on specification of syntactical service interfaces, whereas we consider it important to include abstract behavioral specifications and attributes such as dependencies or security requirements as early as possible. First concepts of such service models are presented in [10, 11, 12]. To our knowledge, no development processes are available yet based on this extended concept of services. First ideas that inspired the presented process are also given in [12].

In future, we plan to extend the given service-based approach in different directions, e.g. by more fine-grained ways to define relationships between services leading to more comprehensive logical service architecture models. In addition, we will look both at the problem of modeling quality of service attributes and of incorporating service contexts (such as the current speed of the car) for context-adaptive services. We also plan to develop a generic security model for service-based systems in the automotive domain.

References

[1] MEWADIS website at `http://www4.in.tum.de/~mewadis`. In German.

[2] W. W. Royce. Managing the Development of Large Software Systems. In *Proceedings of the Ninth International Conference on Software Engineering*, pages 328–338. IEEE, 1987.

[3] Ivar Jacobson, Grady Booch, and James Rumbaugh. *The Unified Software Development Process*. Addison Wesley Longman, Inc., 1999.

[4] D. F. D'Souza and A. C. Wills. *Objects, Components, and Frameworks With UML: The Catalysis Approach*. Addison Wesley Publishing Company, 1998.

[5] Philippe Kruchten. *The Rational Unified Process: An Introduction, Second Edition*. Addison Wesley Longman, Inc., 2000.

[6] Ivar Jacobson, Magnus Christerson, Patrik Jonsson, and Gunnar Övergaard. *Object-Oriented Software Engineering: A Use-Case Driven Approach*. Addison Wesley Longman, Inc., 1992.

[7] Ruth Breu. An Integrated Approach to Use Case Based Development, 2004. To appear.

[8] Ruth Breu. *Objektorientierter Softwareentwurf – Integration mit UML*. Springer-Verlag, 2001. In German.

[9] Ruth Breu, Klaus Burger, Michael Hafner, Jan Jürjens, Gerhard Popp, Guido Wimmel, and Volkmar Lotz. Key Issues of a Formally Based Process Model for Security Engineering. In *Proceedings of the 16th International Conference on Software & Systems Engineering and their Applications (ICSSEA03), Paris, December 2–4, 2003*, 2003.

[10] Manfred Broy. Modeling Services and Layered Architectures. In H. König, M. Heiner, and A. Wolisz, editors, *Formal Techniques for Networked and Distributed Systems*, volume 2767 of *Lecture Notes in Computer Science*, pages 48–61. Springer, 2003.

[11] Manfred Broy. Multi-view Modeling of Software Systems, 2003. Keynote. FM2003 Satellite Workshop on Formal Aspects of Component Software, 8–9 September, Pisa, Italy.

[12] Bernhard Schätz and Christian Salzmann. Service-Based Systems Engineering: Consistent Combination of Services. In *Proceedings of ICFEM 2003, Fifth International Conference on Formal Engineering Methods*. Springer, 2003.

[13] F. Huber, S. Molterer, A. Rausch, B. Schätz, M. Sihling, and O. Slotosch. Tool supported Specification and Simulation of Distributed Systems. In *International Symposium on Software Engineering for Parallel and Distributed Systems*, pages 155–164, 1998.

[14] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, Boston, 1993.

[15] Christian Salzmann. *Modellbasierter Entwurf spontaner Komponentensysteme*. PhD thesis, TU München, 2002. In German.

[16] Open Services Gateway Inititative. OSGi^{TM} Service Platform Specification. Release 3, March 2003, http://www.osgi.org.