Model-Driven Development of Context-Aware Services

João Paulo A. Almeida[1,2], Maria-Eugenia Iacob[1], Henk Jonkers[1], Dick Quartel[2]

[1]Telematica Instituut, P.O. Box 589, 7500 AN Enschede, The Netherlands
e-mail: { JoaoPaulo.Almeida, Maria-Eugenia.Iacob, Henk.Jonkers } @telin.nl

[2]Centre for Telematics and Information Technology, University of Twente,
P.O. Box 217, 7500AE, Enschede, The Netherlands
e-mail: quartel@cs.utwente.nl

Abstract. In this paper, we define a model-driven design trajectory for context-aware services consisting of three levels of models with different degrees of abstraction and platform independence. The models at the highest level of platform independence describe the behaviour of a context-aware service and its environment from an integrated perspective. The models at the intermediate level describe abstract components, which realize the context-aware service in terms of a service-oriented abstract platform. At the lowest level, the realization of a context-aware service is described in terms of specific target technologies, such as Web Services, BPEL and Parlay technologies. Our approach allows service designers to concentrate their efforts on the services they intend to create and offer, by facilitating the handling of context information and automating design steps through model transformation. In addition, our approach enables the reuse of platform-independent models for different target platforms.

1 Introduction

The last few decades have led to an explosion of different means of communication and the availability of ubiquitous (mobile) computing devices and sensors. This combination has enabled the creation of mobile context-aware services, which sense the users' environment to provide relevant functionality to their users. The design and provisioning of such mobile context-aware services is a challenging task, which has justified the development of novel methods, abstractions and infrastructures for the development of such services (e.g., [7, 8, 11, 20]). In addition, the complexity, diversity and fast-changing nature of enabling technology platforms require design approaches that shield designers and providers from platform-specific details allowing them to concentrate their efforts on the services they intend to create and offer. These factors have led us to propose the model-driven design trajectory addressed in this paper.

Our model-driven design approach has three main objectives: (1) to facilitate service design by providing abstractions for context-aware service specification; (2) to improve the reusability of service specifications and designs, by promoting independence from specific technology platforms; and (3) to improve the overall efficiency of the service design process, by promoting the automation of design steps by model transformations. The target platforms we consider include middleware platforms and a part of the mobile telecommunications infrastructure, which is used to send messages to mobile terminal users, to establish calls, and to determine the current location and availability (or presence) of mobile terminal users.

We define three levels of models with different degrees of abstraction and platform independence. The models at the highest level of platform independence describe the behaviour of a context-aware service and its environment from an integrated perspective. This level abstracts from the way context information is obtained, focusing on context-aware behaviour. The models at the intermediate level describe abstract components, which realize the context-aware service in terms of a service-oriented abstract platform. This abstract platform is denoted as the A-MUSE Service Platform (in the Freeband A-MUSE project [12]). The A-MUSE Service Platform provides an abstraction of middleware and service discovery platforms and includes context and action services that are provided by telecom platforms such as Parlay [29]. In addition, this abstract platform supports service discovery with dynamic service properties, which allows one to discover services based on context information. At the platform-specific level, the realization of a context-aware service is described in terms of specific target technologies, such as Web Services, BPEL and Parlay technologies.

The paper is organised as follows. Section 2 sets the theoretical background for our method. A number of concepts such as platform independence and abstract platform are discussed here. Section 3 presents an overview of the different levels of models, abstract platforms and model transformations that play an essential role in the design trajectory. Section 4 discusses the specification of services at the highest level of platform independence in further detail. Section 5 discusses the design of services at the intermediate level of platform independence, and defines the A-MUSE Service Platform. Section 6 describes a model transformation that derives a platform-independent service design from a service specification. Finally, Section 7 summarises our results and indicate future research. The approach is illustrated in this paper with a running example: the Telemonitoring service.

2 Model-Driven Development

In most traditional development practices, the ultimate product of the design process is “the realization”, deployed on available realization platforms. In several model-driven approaches, however, intermediate models are reusable and are considered final products of the design process. These models are carefully defined so as to abstract from details in platform technologies, and are therefore called *platform-independent models* (PIMs), in line with OMG’s Model-Driven Architecture (MDA) [18, 22, 28]. PIMs can be defined with different degrees of platform independence, with respect to the extent to which these models constrain the selection of a target platform. For this reason, we organize the various models of an application into different *levels of platform independence* [3].

The concept of *abstract platform* [3, 4] is an important architectural concept of our approach to model-driven design. An abstract platform is an abstraction of infrastructure characteristics assumed to exist in the construction of platform-independent models of an application at some point in the design process.

An abstract platform defines an acceptable or, to some extent, ideal platform from an application developer's point of view. The characteristics of an abstract platform must have proper mappings onto the set of (concrete) target platforms that are considered for a design. In this way, the notion of abstract platform allows a designer to explicitly define levels of platform independence.

We follow a design process [5, 13] that covers two main phases: the *preparation phase* and the *service creation phase*, both briefly described below.

In the *preparation phase*, experts identify (and, when necessary, define) the required levels of models, their abstract platforms and the modelling language(s) to be used. In addition, during the preparation phase an expert may identify or define (automated) transformations between related levels of models. Since the design trajectory is effectively defined in this phase, it requires careful consideration of application domain requirements, target platform characteristics and design goals.

The results of the preparation phase are used in the service creation phase, as illustrated schematically in Figure 1.

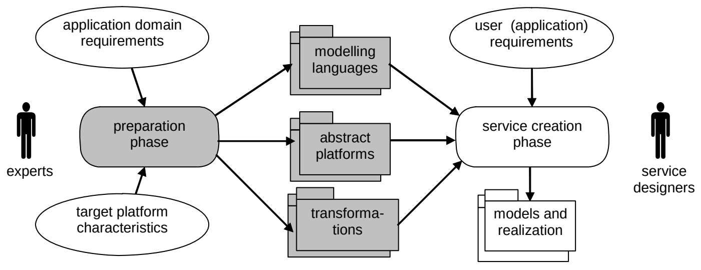


Figure 1. The preparation phase and its results

The design process described in [5, 13] is neutral with respect to specific application domains and target platforms. In this paper, however, we consider the specific case of context-aware services, which are ultimately deployed on top of a (telecommunications) services infrastructure and middleware platforms. In this case, our objective in the preparation phase is to capture design knowledge that is applicable to a large number of different context-aware services and that can be later reused in the service creation phase in the design of a specific service, which addresses specific service requirements. This includes knowledge on how to cope with distribution in the middleware platforms targeted, but also includes knowledge on how context information is handled in the target context-aware services infrastructure.

The *service creation phase* entails the creation of models of a specific service using specific modelling languages and abstract platforms and applying (manual and automated) transformations to models. The service creation phase leads ultimately to a realization (or alternative realizations) of the service that satisfies user requirements, while capturing reusable platform-independent models of the service design. This phase also entails analysis, testing and validation of models and realizations. For an extensive presentation of the methodological support for both the preparation and service creation phase we refer to [5].

3 Design Trajectory Overview

This section explores the main activities and deliverables of the preparation phase in the design trajectory for context-aware services. We first consider a generic decomposition (architecture) of a context service. Based on this decomposition, we identify the characteristics of the A-MUSE Service Platform, and derive the necessary levels of models to be used in the service creation phase.

3.1 Context-Aware Services and the A-MUSE Service Platform

Context-awareness refers to the capabilities of applications to provide relevant services to their users by sensing and exploring the users' context [7, 11, 20]. *Context* is defined as a "collection of interrelated conditions in which something exists or occurs" [11]. The users' context often consists of a collection of conditions, such as, e.g., the users' location, environmental aspects (temperature, light intensity, etc.) and activities [8]. The users' context may change dynamically, and, therefore, a basic requirement for a context-aware system is its ability to sense context and to react to context changes (without intervention of the user). Changes in context can be considered external stimuli, namely *events*, which require a (re)*action* from the context-aware system.

A decomposition of a context-aware service reveals the architecture shown in Figure 2. This architecture consists of *context sources*, which are able to sense context and represent it as context information in the scope of the system. The service provided by context sources is used by a *coordination component*, which requests actions to be executed by *action providers* depending on situations that can be inferred from context information. For example, two users may require a service to establish a call between them when they are located within a certain range of each other. An example of an action provider suitable for this service is a Parlay gateway [29], which can be requested to establish a telephone call between two users. Each user accesses the service through a user component, which provides the user interface and interacts with the *coordination component*.

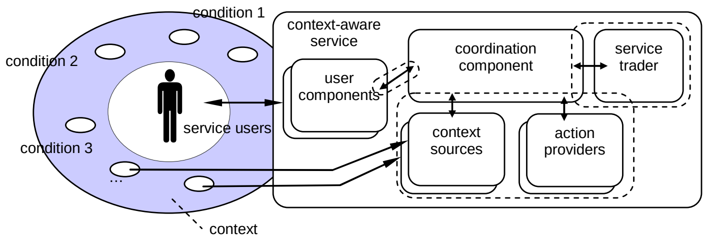


Figure 2. Decomposition of a context-aware service

The user components and the coordination component exhibit service-specific behaviour, and are called service components. In contrast, context sources and action providers are general-purpose and, therefore, can be reused in several different context-aware services. For this reason, we consider context sources and action providers as part of the *A-MUSE Service Platform* (see elements encircled with dashed lines in Figure 2). This platform also supports the interaction between the *user components* and the *coordination component* and the interactions between the coordination component and context sources and action providers. The service provided by context sources and action providers to the coordination component is registered in a *service trader*. This allows the coordination component to select context sources and action providers dynamically according to service offers that are registered in the service trader. Service offers have properties that can be used to select a particular service offer. For example, an action provider can be selected according to its geographical proximity to a user.

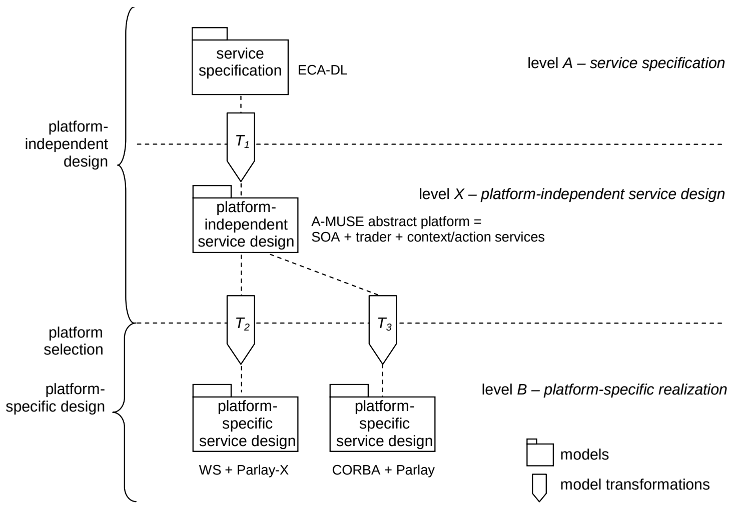
3.2 Levels of Models for Context-Aware Services Development

We define the scope of the design trajectory to include the design activities from the specification of a service at a high-level of abstraction to the realization of this service. Given this scope, one extreme approach to organizing the design trajectory would be to have one level of service specification and one level of service realization and one transformation that relates these two levels. However, the gap between these two levels of models may be very large. This means that a lot of effort should be invested in defining the transformation. This effort is rendered useless when changes in the target platform invalidate the transformation. Therefore, the opportunities for reuse can be increased if an intermediate level of models is introduced. This level of models uses an abstract platform to achieve platform independence, and, hence, models at this level can be reused for different target platforms. The organization of the design trajectory is depicted in Figure 3. The three levels of models we have identified are:

Service specification level. This level of models describes the behaviour of a context-aware service from an external perspective. At this level, we do not distinguish the environment (including service users) and the service provider. The concept of action is used to model both the occurrence of events originated from context sources and the execution of actions. This allows modelling context-aware behaviour at a high-level of abstraction. At this level of abstraction, the service specifier ignores how context information is obtained from context sources. Services are described in a domain-specific language called Events-Conditions-Actions Domain Language (ECA-DL).

Platform-independent service design level. This level of models describes the behaviour of a context-aware service from an internal perspective, revealing a service-specific coordination component and the A-MUSE Service Platform. The A-MUSE Service Platform is the result of the composition of: a Service-Oriented-Architecture (SOA) abstract platform, which uses abstract interactions [2] to support the communication of application parts in this design; a service discovery platform which consists of a service trader; and general-purpose context and action services. This level of models reveals how context and action services are registered, searched for, and used by coordination components. The transformation denoted with T_1 in Figure 3 introduces the coordination component so that the behaviour of the composition of the coordination component and the A-MUSE Service Platform performs the service specified at the service specification level.

Platform-specific service design level. This level of models describes the realization of the service for particular platforms. The flexibility of the relation between the platform-independent service design level and the platform-specific service design level allows different middleware platforms to be used. Model transformations can be used to create models at this level. For example, one could use them to generate the BPEL specification of the context-aware service that orchestrates (using a BPEL engine and SOAP [30]) web services (e.g. Parlay-X services [29]) for which WSDL interfaces [31] are provided. This transformation is illustrated in Figure 3 denoted by T_2. In this figure, T_3 denotes a transformation to CORBA and Parlay.



service specification
ECA-DL
level *A – service specification*
platform-independent design
T_1
level *X – platform-independent service design*
platform-independent service design
A-MUSE abstract platform = SOA + trader + context/action services
platform selection
T_2
T_3
platform-specific design
level *B – platform-specific realization*
platform-specific service design
platform-specific service design
WS + Parlay-X
CORBA + Parlay
models
model transformations

Figure 3. Design trajectory consisting of three levels of models

4 Service Specification Level

At the level of service specification a context-aware service can be described in terms of *events*, which represent contextual changes, *queries* to context sources, and *actions*, which represent actions to be performed in order to provide the service to the user. We defined this level through a domain-specific language for the domain of context-aware services specification. We specialize elements of a general-purpose design language, namely the Interaction System Design Language (ISDL) [15, 26, 27], thus defining a dialect of it, which we call Events-Conditions-Actions Domain Language (ECA-DL). This language provides a means to specify behaviours in terms of actions and causality relations between these actions. The specialization consists of defining special types of actions, namely, context events (CE), context query requests (CQ), context query responses (CQ') and action invocation requests (AI) and action invocation responses (AI'). Context query requests and context query responses are always related by causality, forming a pattern. The definition of the ECA-DL is illustrated schematically in Figure 4 (complete meta-models for ECA-DL in OMG's Meta-Object Facility (MOF) are described in [6]).

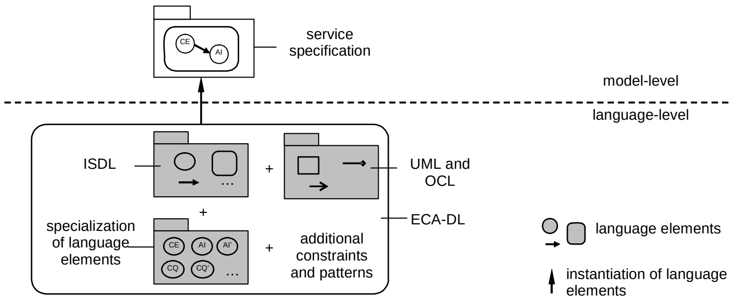

service specification
CE
AI
model-level
language-level
ISDL
+
UML and OCL
+
ECA-DL
specialization of language elements
CE
AI
AI'
CQ
CQ'
+
additional constraints and patterns
language elements
instantiation of language elements

Figure 4. Definition of the ECA-DL language for context-aware service specification

In order to illustrate the usage of the proposed language and approach, we consider the design of a "Telemonitoring service" for epilepsy patients [17]. The service assumes the availability of sensor technology that enables a wearable 24-hour seizure monitoring system. A couple of minutes before the onset of a seizure, the monitoring system detects its signs. The patient is warned of an imminent seizure and based on location information a voluntary aid person (e.g., spouse) or a health team can be dispatched for assistance.

The Telemonitoring service specification is depicted in Figure 5. Ovals represent specialized actions (with a naming convention with suffixes). Arrows indicate enabling relations between actions; white diamonds represent choice and white squares denote disjunction.

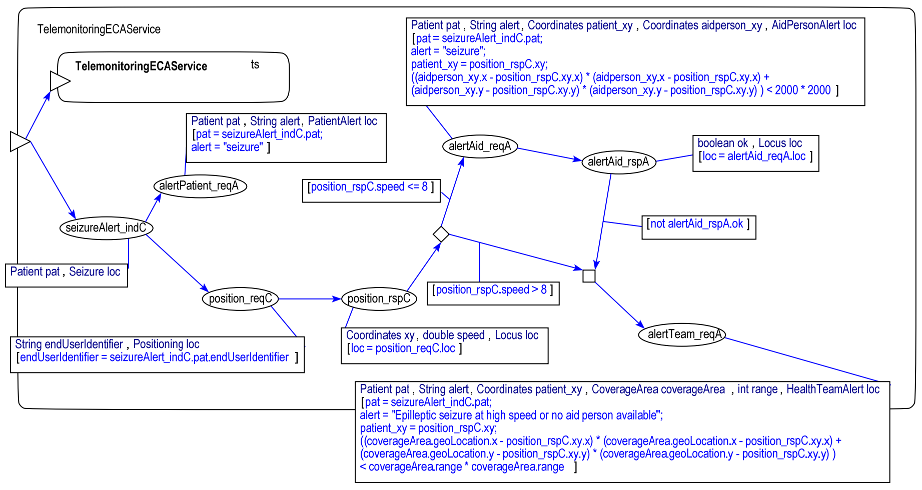

TelemonitoringECAService
TelemonitoringECAService
ts
Patient pat , String alert , Coordinates patient_xy , Coordinates aidperson_xy , AidPersonAlert loc
[pat = seizureAlert_indC.pat;
alert = "seizure";
patient_xy = position_rspC.xy;
((aidperson_xy.x - position_rspC.xy.x) * (aidperson_xy.x - position_rspC.xy.x) +
(aidperson_xy.y - position_rspC.xy.y) * (aidperson_xy.y - position_rspC.xy.y)) < 2000 * 2000]
Patient pat , String alert , PatientAlert loc
[pat = seizureAlert_indC.pat;
alert = "seizure"]
alertAid_reqA
alertAid_rspA
boolean ok , Locus loc
[loc = alertAid_reqA.loc]
alertPatient_reqA
[position_rspC.speed <= 8]
seizureAlert_indC
[not alertAid_rspA.ok]
Patient pat , Seizure loc
position_reqC
position_rspC
[position_rspC.speed > 8]
alertTeam_reqA
Coordinates xy , double speed , Locus loc
[loc = position_reqC.loc]
String endUserIdentifier , Positioning loc
[endUserIdentifier = seizureAlert_indC.pat.endUserIdentifier]
Patient pat , String alert , Coordinates patient_xy , CoverageArea coverageArea , int range , HealthTeamAlert loc
[pat = seizureAlert_indC.pat;
alert = "Epilleptic seizure at high speed or no aid person available";
patient_xy = position_rspC.xy;
((coverageArea.geoLocation.x - position_rspC.xy.x) * (coverageArea.geoLocation.x - position_rspC.xy.x) +
(coverageArea.geoLocation.y - position_rspC.xy.y) * (coverageArea.geoLocation.y - position_rspC.xy.y))
< coverageArea.range * coverageArea.range]

Figure 5. The Telemonitoring service specification (exported from Grizzle [14] ISDL tool)

A simple naming convention has been used to indicate the type of action: suffix _indC denotes context events; suffixes _reqC and _rspC denote context query requests and context query responses; and suffixes _reqA and _rspA denote action invocation requests and action invocation responses. The event seizureAlert_indC represents that an (imminent) epileptic seizure has been detected in a patient being monitored. The action alertPatient_reqA requests the patient to be informed about the seizure. Following a seizure alert, the patient's current location and speed is requested (position_reqC followed by position_rspC). An aid person within range of the patient is informed of the seizure and the current location of the patient (alertAid_reqA). When no aid persons are available or the speed of the patient exceeds a certain value (which could indicate a hazardous situation) a health team capable of handling epileptic seizures is dispatched to the location of the patient. The Grizzle ISDL tool [14] is used for model editing and simulation of service specifications.

ISDL allows designers to use a modelling language of their choice to define the attributes of actions and constraints on these attributes. For ECA-DL, we have chosen to use UML class diagrams [25] for the (context) information attributes. Further, we use a subset of the Object Constraint Language (OCL) [24] to express constraints on information attributes. Constraints on information attributes serve to specify context-dependent conditions and action results, and can also be used to specify required *properties* of action services. This is illustrated in the constraints of action alertAid_reqA in Figure 5: only an aid person within range of the patient is informed of the seizure.

5 Platform-Independent Service Design Level

At the platform-independent service design level, the service is provided by a service-specific coordination component in cooperation with the A-MUSE Service Platform. This abstract platform is the result of the composition of: a Service-Oriented-Architecture (SOA) abstract platform; a service discovery platform; and general-purpose context and action services. The structure of platform-independent service designs is depicted schematically in Figure 6, revealing the hierarchy of elements that constitute the A-MUSE Abstract Platform. This figure also shows the relation between the service specification level and the platform-independent service design level.

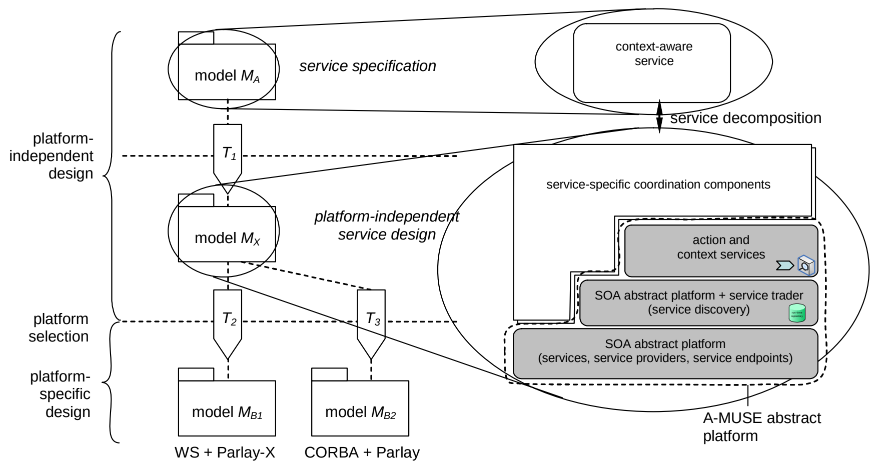

platform-independent design
platform selection
platform-specific design
model M_A
service specification
context-aware service
service decomposition
T_1
model M_X
platform-independent service design
service-specific coordination components
action and context services
SOA abstract platform + service trader (service discovery)
SOA abstract platform (services, service providers, service endpoints)
T_2
T_3
model M_{B1}
model M_{B2}
WS + Parlay-X
CORBA + Parlay
A-MUSE abstract platform

Figure 6. Abstract platforms at the platform-independent service design level

A schematic overview of the approach for the definition of the hierarchy of abstract platforms that constitutes the A-MUSE Service Platform is shown in Figure 7. The *service-oriented abstract platform* is defined using a pure language-level approach [4], i.e., the modelling language used defines the characteristics of the abstract platform. The language adopted is ISDL (meta-models for ISDL in MOF are described in [6], based on [9]). The information and location attributes of actions are described with UML. Constraints on these attributes are described with OCL. Since this level defines a composition of various (potentially distributed) components, which operates through services, it is necessary to describe the interactions between components. This is done with abstract interactions, which can be represented in ISDL ([2] discusses how these abstract interactions can be realized on different middleware platforms). The *service discovery abstract platform* is built on top of the underlying service-oriented abstract platform and is defined with a model-level approach, i.e., with the definition of reusable modelling artefacts. This abstract platform consists of a service trader component, defined in ISDL. On top of that, context and action services are defined, completing the A-MUSE Service Platform.

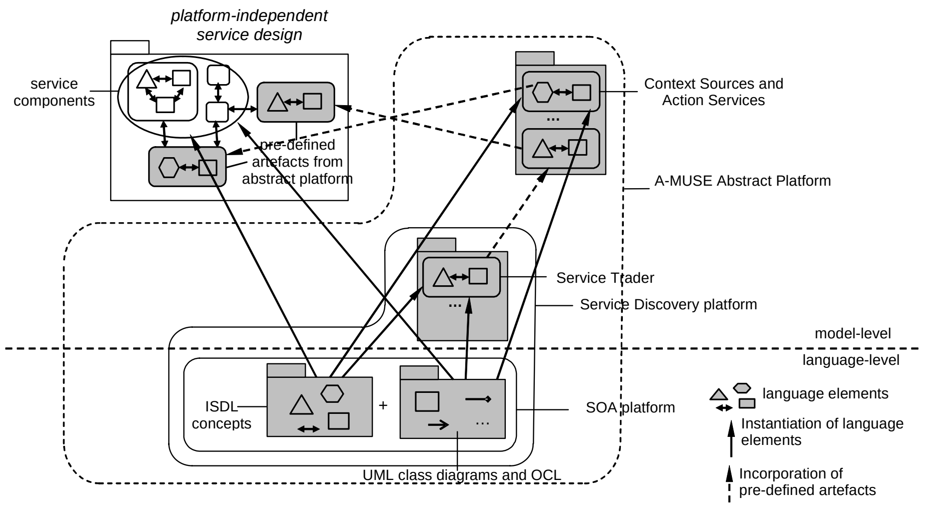

platform-independent service design
service components
pre-defined artefacts from abstract platform
Context Sources and Action Services
A-MUSE Abstract Platform
Service Trader
Service Discovery platform
model-level
language-level
ISDL concepts
+
SOA platform
UML class diagrams and OCL
language elements
Instantiation of language elements
Incorporation of pre-defined artefacts

Figure 7. Defining the hierarchy of abstract platforms definition

We omit any detailed ISDL descriptions of the service trader and context and action services due to space limitations. We refer the reader to [6] for the complete ISDL specifications with OCL constraints and UML class diagrams for information attributes.

6 Model transformation

Given a service specification in ECA-DL, a platform-independent service design, specified in standard ISDL, can be derived automatically using model transformation. As a proof of concept, we have implemented this transformation using the Graph Rewriting And Transformation (GReAT) software developed at Vanderbilt University

[1, 18]. GReAT has been implemented within the Generic Modelling Environment (GME) [19], a configurable toolset for the creation of domain-specific modelling environments. An editor for a domain-specific language (called a 'paradigm' in GME) can be created based on a metamodel of the language specified in MetaGME, a graphical UML-like metamodelling language (which in itself has been defined as a GME paradigm) [18]. One of the main drawbacks of the GME is its use of proprietary formats for metamodelling and model exchange, rather than conforming to standards such as MOF and XMI.

In GReAT, model transformations are specified using a graphical graph transformation language called UML Model Transformer (UMT), which has also been defined as a GME paradigm. The transformation specification makes use of metamodels of the source and destination languages defined in MetaGME. For our example, we have defined metamodels for ECA-DL (source) and ISDL (target), and a UMT specification to derive a platform-independent service design from a service specification. Figure 8 illustrates this.

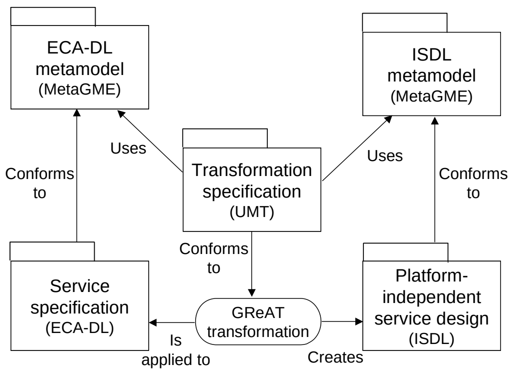

ECA-DL metamodel (MetaGME)
ISDL metamodel (MetaGME)
Uses
Uses
Conforms to
Transformation specification (UMT)
Conforms to
Conforms to
Service specification (ECA-DL)
GReAT transformation
Platform-independent service design (ISDL)
Is applied to
Creates

Figure 8. Overview of the transformation approach

One of the central concepts of the GReAT model transformation approach is the substitution of graph patterns, which provides an intuitive way to express the types of transformations that we want to perform here. Figure 9 shows an example of a UMT transformation rule, which for each ECA-DL action of type AI (action invocation request), creates a sequence of three interactions in the ISDL design. These are interactions between the coordination service component and the A-MUSE abstract platform.

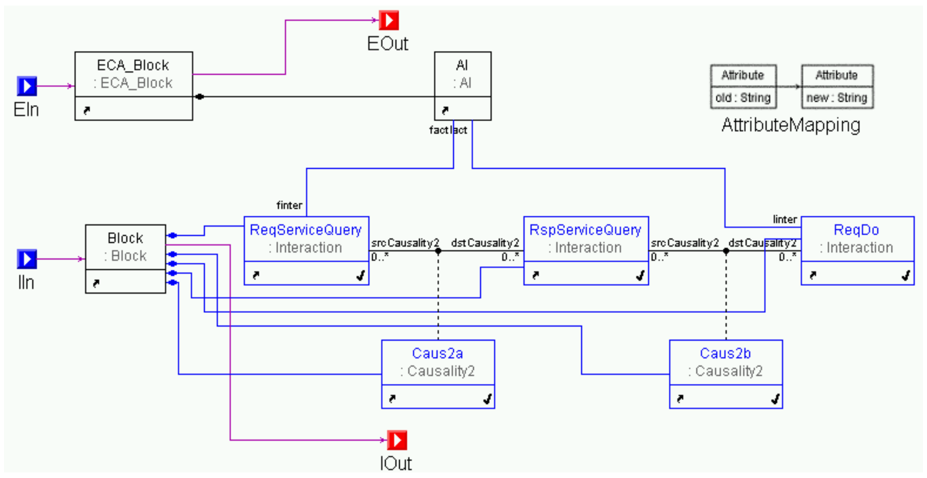


Figure 9. Example of a UMT transformation rule

The interactions realize the abstract action, involving a request to the service trader, a response from the service trader and the invocation of the appropriate action service according to the response issued by the service trader. Similarly, rules have been defined for the other ECA-DL action types, as well as rules to derive the relations between actions and rules concerning the action attributes. Figure 10 shows the effect of this rule in an informal way.

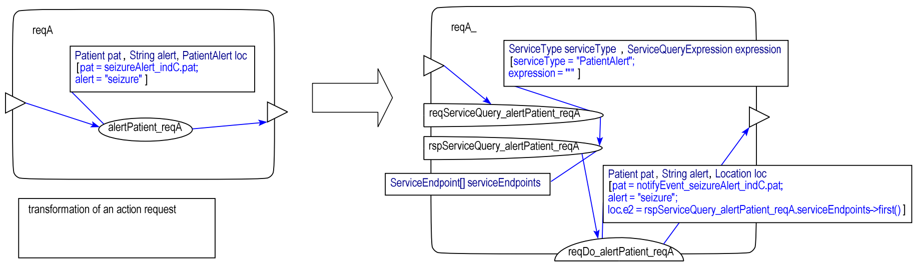
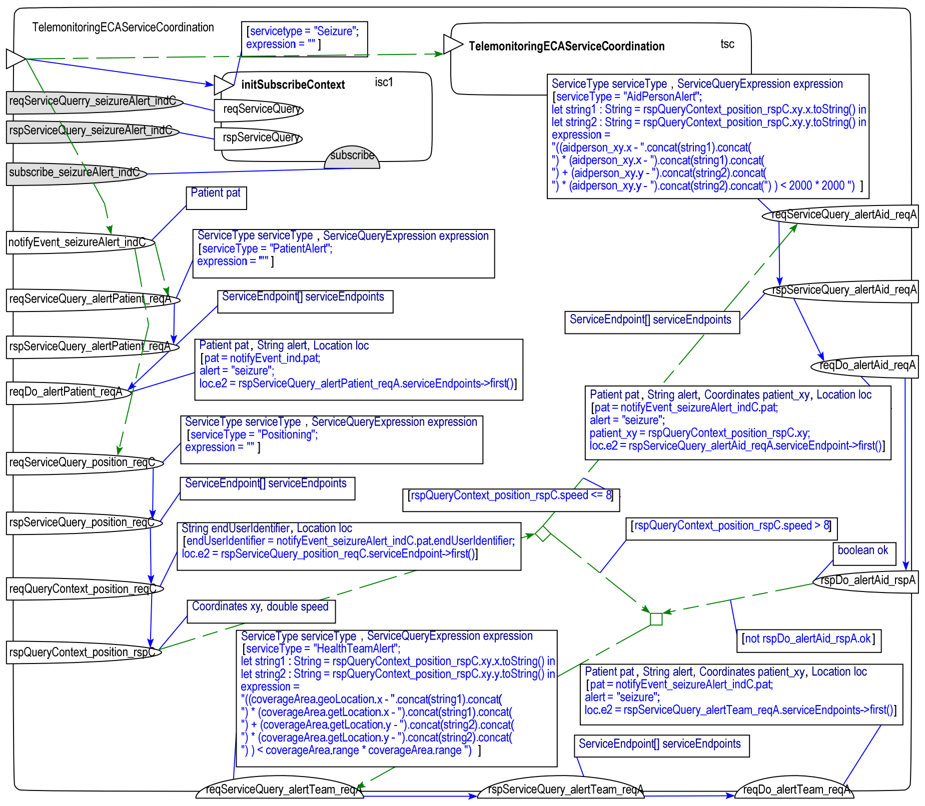

reqA
Patient pat , String alert, PatientAlert loc
[pat = seizureAlert_indC.pat;
alert = "seizure"]
alertPatient_reqA
transformation of an action request
reqA_
ServiceType serviceType , ServiceQueryExpression expression
[serviceType = "PatientAlert";
expression = ""]
reqServiceQuery_alertPatient_reqA
rspServiceQuery_alertPatient_reqA
ServiceEndpoint[] serviceEndpoints
Patient pat , String alert, Location loc
[pat = notifyEvent_seizureAlert_indC.pat;
alert = "seizure";
loc.e2 = rspServiceQuery_alertPatient_reqA.serviceEndpoints->first()]
reqDo_alertPatient_reqA

Figure 10. Informal illustration of the AI transformation rule

The platform-independent service design is the result of the application of all the transformation rules to the service specification. Figure 11 shows the generated coordination component. The dashed lines represent causality relations already present in the service specifications.



TelemonitoringECAServiceCoordination
[servicetype = "Seizure";
expression = ""]
TelemonitoringECAServiceCoordination
tsc
initSubscribeContext
isc1
reqServiceQuery
rspServiceQuery
subscribe
reqServiceQuerry_seizureAlert_indC
rspServiceQuery_seizureAlert_indC
subscribe_seizureAlert_indC
ServiceType serviceType , ServiceQueryExpression expression
[serviceType = "AidPersonAlert";
let string1 : String = rspQueryContext_position_rspC.xy.x.toString() in
let string2 : String = rspQueryContext_position_rspC.xy.y.toString() in
expression =
"((aidperson_xy.x - ".concat(string1).concat(
") * (aidperson_xy.x - ").concat(string1).concat(
") + (aidperson_xy.y - ").concat(string2).concat(
") * (aidperson_xy.y - ").concat(string2).concat(")) < 2000 * 2000 ")]
Patient pat
notifyEvent_seizureAlert_indC
ServiceType serviceType , ServiceQueryExpression expression
[serviceType = "PatientAlert";
expression = ""]
reqServiceQuery_alertAid_reqA
rspServiceQuery_alertAid_reqA
reqServiceQuery_alertPatient_reqA
ServiceEndpoint[] serviceEndpoints
ServiceEndpoint[] serviceEndpoints
rspServiceQuery_alertPatient_reqA
Patient pat , String alert, Location loc
[pat = notifyEvent_ind.pat;
alert = "seizure";
loc.e2 = rspServiceQuery_alertPatient_reqA.serviceEndpoints->first()]
reqDo_alertAid_reqA
reqDo_alertPatient_reqA
Patient pat , String alert, Coordinates patient_xy, Location loc
[pat = notifyEvent_seizureAlert_indC.pat;
alert = "seizure";
patient_xy = rspQueryContext_position_rspC.xy;
loc.e2 = rspServiceQuery_alertAid_reqA.serviceEndpoint->first()]
ServiceType serviceType , ServiceQueryExpression expression
[serviceType = "Positioning";
expression = ""]
reqServiceQuery_position_reqC
ServiceEndpoint[] serviceEndpoints
[rspQueryContext_position_rspC.speed <= 8]
rspServiceQuery_position_reqC
String endUserIdentifier, Location loc
[endUserIdentifier = notifyEvent_seizureAlert_indC.pat.endUserIdentifier;
loc.e2 = rspServiceQuery_position_reqC.serviceEndpoint->first()]
[rspQueryContext_position_rspC.speed > 8]
boolean ok
reqQueryContext_position_reqC
rspDo_alertAid_rspA
Coordinates xy, double speed
rspQueryContext_position_rspC
ServiceType serviceType , ServiceQueryExpression expression
[serviceType = "HealthTeamAlert";
let string1 : String = rspQueryContext_position_rspC.xy.x.toString() in
let string2 : String = rspQueryContext_position_rspC.xy.y.toString() in
expression =
"((coverageArea.geoLocation.x - ".concat(string1).concat(
") * (coverageArea.getLocation.x - ").concat(string1).concat(
") + (coverageArea.getLocation.y - ").concat(string2).concat(
") * (coverageArea.getLocation.y - ").concat(string2).concat(
")) < coverageArea.range * coverageArea.range ")]
[not rspDo_alertAid_rspA.ok]
Patient pat , String alert, Coordinates patient_xy, Location loc
[pat = notifyEvent_seizureAlert_indC.pat;
alert = "seizure";
loc.e2 = rspServiceQuery_alertTeam_reqA.serviceEndpoints->first()]
ServiceEndpoint[] serviceEndpoints
reqServiceQuery_alertTeam_reqA
rspServiceQuery_alertTeam_reqA
reqDo_alertTeam_reqA

Figure 11. Coordination component for Telemonitoring service (exported from Grizzle)

The TelemonitoringECAServiceCoordination enforces the behaviour defined at the service specification level (shown in Figure 5). The coordination component uses context and action services that constitute the A-MUSE Service Platform, including the ability to send and receive SMSs and to check the position and availability of mobile terminal users. The service trader is consulted to find appropriate context sources and action services depending on the constraints on information attributes that have been specified at the service specification level. For example, the seizureAlert_indC context event is refined in a number of interactions that lead to the notifyEvent_SeizureAlert_indC between the TelemonitoringECAServiceCoordination and the EventBasedSeizureService. The constraint on the location of aid persons in alertAid_reqA has been transformed into a constraint on the value of a service property in the query of the reqServiceQuery_alertAid_reqA interaction. This is a dynamic service property that is evaluated by the service trader after the query is issued.

7 Conclusions

In this paper we have proposed a model-driven design trajectory for context-aware and mobile services, in which a number of concepts such as platform independence, abstract platform, context-awareness and service orientation play an important role. We have presented the design trajectory by discussing the necessary levels of models, the choice of modelling languages, and the definition of platforms and transformations. Further, we have illustrated the application of our approach by means of an example (i.e., the Telemonitoring service). The Telemonitoring design exercise helped us to emphasize the role of model transformations, but also to understand to what extent the whole design process can be automated.

The service specification level emphasizes ease of use for the service specifier and platform independence for service specifications. A context-aware service is defined from its integrated perspective abstracting from any components that may support the execution of the service in terms of technology platforms such as Parlay or Parlay-X (which provide context and action services in the telecommunications domain) and Web Services or CORBA (which provide service-oriented middleware architectures, including some service discovery functionality).

The abstract platform at the platform-independent service design level has been chosen based on the pattern of service discovery found in a number of middleware platforms (e.g., OMG CORBA trader [23] and the UDDI registry [21]) and in the ODP trader [16]. The trader service in the A-MUSE Service Platform is capable of supporting a simple constraint language and is capable of supporting dynamic service properties, which allows contextual information to be used to trade for services, as we have shown in the Telemonitoring example. These capabilities of the service trader do not have to be implemented in the coordination component, therefore simplifying the design of transformations that use the A-MUSE platform as target. For a discussion on the realization of the service trader in UDDI and CORBA trader we refer the reader to [6]. We believe the service discovery abstract platform described in this paper is domain neutral and can be used where a service-oriented architecture is needed, without dependence on a particular technology platform such as Web Services.

We have used ISDL (and ECA-DL as a specialization thereof) to model the behavioural aspects of services for three main reasons. Firstly, ISDL supports a broad spectrum of abstraction levels which allows us to cover from service specification to service design seamlessly. Secondly, the concept of abstract interaction enables us to capture service designs in a middleware-platform-independent manner (as shown in [2]). And, finally, conformance rules have been defined [26] which can be used to verify whether service designs respect service specifications.

We have used UML class definitions and OCL constraints to model context information. In the context of the A-MUSE project, we are investigating the use of semantic models expressed in OWL. The latter may allow the designer to automatically reason whether, for example, two services are semantically connectible. We are also working on the further development of the ECA-DL and the A-MUSE abstract platform. Tool support for the various levels of models in this design trajectory will be incorporated in an integrated environment for model-driven service engineering.

Acknowledgements

This work is part of the Freeband A-MUSE project (http://a-muse.freeband.nl), which is sponsored by the Dutch government under contract BSIK 03025. Marten van Sinderen, Luís Ferreira Pires and Remco Dijkman are acknowledged for their suggestions and remarks on the model-driven approach reported in Sections 2 and 3.

References

1. Agrawal, A., Karsai, G., Ledeczi, A.: An end-to-end domain-driven software development framework. In: Companion of the 18th Annual ACM SIGPLAN Conf. on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA), ACM Press (2003) 8–15
2. Almeida, J.P.A., Dijkman, R., Ferreira Pires, L., Quartel, D., van Sinderen, M.: Abstract Interactions and Interaction Refinement in Model-Driven Design. In: Proc. 9th IEEE EDOC Conference (EDOC 2005), IEEE Computer Society Press (2005) 273–286
3. Almeida, J.P.A., van Sinderen, M., Ferreira Pires, L., Quartel, D.: A systematic approach to platform-independent design based on the service concept. In: Proc. 7th IEEE Int'l Conf. on Enterprise Distributed Object Computing (EDOC 2003). IEEE Computer Society Press (2003) 112–123
4. Almeida, J.P.A. Dijkman, R. van Sinderen, M., Ferreira Pires, L.: On the Notion of Abstract Platform in MDA Development, In: Proc. 8th IEEE Int'l Conf. on Enterprise Distributed Object Computing (EDOC 2004), IEEE Computer Society Press (2004) 253–263
5. Almeida, J.P.A., Iacob, M.E., Iacob, S.: Methodological Framework for Freeband Services Development, Freeband A-MUSE/D2.3a, TI/RS/2004/092, Telematica Instituut, Enschede, The Netherlands (2004); https://doc.telin.nl/dscgi/ds.py/Get/File-47390
6. Almeida, J.P.A., Iacob, M.E., Jonkers, H., Quartel, D.: Platform-Independent Modelling of Service Infrastructure Components, Freeband A-MUSE/D1.6, TI/RS/2005/078, Telematica Instituut, Enschede, The Netherlands (2005); https://doc.telin.nl/dscgi/ds.py/Get/File-59319
7. Dey, A. K., Salber, D., and Abowd, G. D.: A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. Human-Computer Interaction, 16(2-4) (2001) 97–166
8. Chen, H. Finin, T., Joshi, A.: An ontology for context-aware pervasive computing environments, Knowledge Engineering Review, Special Issue on Ontologies for Distributed Systems, Vol. 18, No. 3. Cambridge University Press (2003) 197–207
9. Dijkman, R.M.: Consistency in Multi-Viewpoint Architectural Design, Ph.D. thesis, University of Twente, The Netherlands (2006)
10. Dirgahayu, T.: Model-Driven Engineering of Web Service Compositions: A Transformation from ISDL to BPEL, M.Sc. thesis, University of Twente, The Netherlands (2005)
11. Dockhorn Costa, P. Ferreira Pires, L., van Sinderen, M.: Designing a Configurable Services Platform for Mobile Context-Aware Applications, International Journal of Pervasive Computing and Communications (JPCC), Vol. 1, No. 1. Troubador Publishing (2005)
12. Freeband A-MUSE Project; http://a-muse.freeband.nl
13. Gavras, A., Belaunde, M., Ferreira Pires, L., Almeida, J.P.A.: Towards an MDA-based Development Methodology for Distributed Applications. In: Software Architecture: First European Workshop (EWSA2004), LNCS 3047, Springer (2004) 230–240
14. Grizzle; http://isdl.ctit.utwente.nl/tools/grizzle
15. ISDL home; http://isdl.ctit.utwente.nl/
16. ITU-T / ISO: ODP Trading Function: Specification, ITU-T Recommendation X.950 | IS 13235-1 (1997)

17. Jonkers, H., Iacob, M.E., Lankhorst, M., Strating, P.: Integration and Analysis of Functional and Non-Functional Aspects in Model-Driven E-Service Development. In: Proc. 9th IEEE EDOC Conference (EDOC 2005), IEEE Computer Society Press (2005) 229–238
18. Karsai, G., Agrawal, A.: Graph transformations in OMG's Model Driven Architecture. In: Applications of Graph Transformations with Industrial Relevance, Second International Workshop (AGTIVE2003), Charlottesville, VA, USA (2003) 243–259
19. Ledeczi, A. et al.: The Generic Modeling Environment. In: Proc. Workshop on Intelligent Signal Processing, Budapest, Hungary (2001)
20. McFadden, T., Henricksen, K., Indulska, J., Mascaro, P.: Applying a Disciplined Approach to the Development of a Context-Aware Communication Application. In: 3rd IEEE Int'l Conf. on Pervasive Computing and Communications (PerCom), IEEE Computer Society Press (2005) 300–306
21. OASIS: OASIS - Committees - OASIS UDDI Specifications TC; http://oasis-open.org/committees/uddi-spec/doc/tcspecs.htm
22. Object Management Group: MDA-Guide, Version 1.0.1, omg/03-06-01 (2003)
23. Object Management Group: Trading Object Service Specification, Version 1.0, formal/00-06-27 (2000)
24. Object Management Group: Unified Modelling Language: Object Constraint Language version 2.0, ptc/03-10-04 (2003)
25. Object Management Group: UML 2.0 Superstructure, ptc/03-08-02 (2003)
26. Quartel, D.: Action relations Basic design concepts for behaviour modelling and refinement, Ph.D. thesis, University of Twente, Enschede, The Netherlands (1998)
27. Quartel, D. Ferreira Pires, L., van Sinderen, M.: On Architectural Support for Behaviour Refinement. In: Journal of Integrated Design and Process Science, Vol. 6, No. 1. IOS (2002)
28. Selic, B.: The Pragmatics of Model-Driven Development. IEEE Software, Vol. 20, No. 5, IEEE Computer Society Press (2003) 19–25
29. The Parlay Group: "The Parlay Group – Specifications"; http://www.parlay.org
30. World Wide Web Consortium: SOAP Version 1.2 Part 1: Messaging Framework, W3C Proposed Recommendation (2003); http://www.w3.org/TR/soap12-part1
31. World Wide Web Consortium: Web Services Description Language (WSDL) 1.1, W3C Note (2001); http://www.w3.org/TR/wsdl