UbiCompForAll

Work package:	Date:
WP2 Architecture	June 4th 2010

Partner / author

SINTEF / Richard Sanders (w/Erlend & Jacqueline)

Classification: Public

Memo title::

Architecture: Context view

Memo:

This memo describes the UbiCompForAll architecture. It follows the MAFIIA framework for defining architectures [1]. The Context viewpoint and Requirements viewpoint have been defined, with a Business aspects model, the Environment systems model and the Business to system mapping model. In addition a component view has been sketched. The Business aspects model is exemplified by the City Guide and Reminder applications, the latter being a generalization of the "Doctor's appointment" scenario.

Table of contents

1 Terms and concepts

The following is a list of terms and concepts defined for use in the architecture and by the project.

Software – end-user viewpoint	
User service	Functionality provided to a *service user* or a group of *service users*. User services are created by *service composers*.
Application	A *software component* running on a user's device that provides a specified functionality to a user.
Software – computational viewpoint	
Software component	Software unit that encapsulates a set of related functions. According to Szyperski [2], a component provides functionality through contractually specified interfaces and defines explicit dependencies to its environment such as required interfaces and acceptable execution platform(s).
Software service	Functionality provided by a (one or more) *software component* to other *software components* via (one or more) software interfaces. Services may be provided locally or by a *service host*.
User involvement in software creation	
End-user development (EUD)	Software research field aiming at developing environments that allow *end users* without professional programming background to develop or modify their own software artefacts (e.g. applications, web sites) [3].
End-user software engineering	Research area related to EUD that focuses on software quality and aims at empowering *end users* to create bug-free and robust software applications. UbiCompForAll addresses this research area [4].
End-user service composition	Specific research area within EUD where *end users* develop or modify software artefacts through the assembly of *services*. UbiCompForAll addresses this research area.
Stakeholders roles	
End user	In the role of an *end user* an actor uses an *application* via some user interface. In UbiCompForAll end users include both *service composers* and *service users*. Typically end users have a non-professional programming background
Service composer (composer)	In the role of a *service composer* an actor develops or modifies *user services* using UbiCompForAll *composition notations, methods* and *tools*.
Service user[1]	In the role of a service user an actor participates in *user services*.
Developer	In the role of a *developer* an actor creates or adapts *software components* or *software services* for general use. Typically developers have a professional programming background.
Domain developer	In the role of a *domain developer* an actor creates or adapts *software components* or *services* for use in the UbiCompForAll *service composition framework*. Typically domain developers have a professional programming background.
Environment	In the role of *environment developer* an actor specifies a *service composition*

[1] We distinguish between *primary* and *ordinary* service users, the latter participates without requiring any specific deployment of services or components used by a user service.

developer	*framework* and *runtime environment* for use in UbiCompForAll. Typically environment developers have a professional programming background
Service host	An organization that offers facilities for hosting a) *software services* created by *developers* and/or b) *user services* created by *service composers*.
UbiCompForAll software artefacts	
Service composition framework	Collection of notations, tools and middleware support developed in UbiCompForAll for the composition of user services.
Service composition notations	Notations used to specify a composition of *user services*, including conditions, constraints and non-functional requirements.
Adaptation tool (*UbiCompPRO*)	A software tool that aids the *domain developer* in adapting *software components* or *services* to be used in the UbiCompForAll *service composition framework*.
Service composition tool (*UbiComposer*)	A software tool that aids the *service composer* in composing *user services*.
Middleware support (*UbiCompRun*)	A software infrastructure that is needed to enable the composition, execution and management of *user services*.
Building block	An element used by a *service composer* when composing a *user service* (can represent a *software component* or *software service*)

2 Concerns

Concerns are related to the documentation of the functional aspects of the target system and its environment. Functional aspects that are considered to be of such importance that it should be treated separately and be specifically visible in the documentation should be identified and treated as a concern.

Important concerns for UbiCompForAll are understandability, learnability, operability, installability, and configurability. Understandability, learnability and operability are sub-characteristics of usability.

- Understandability: "*The capability of the software product to enable the user to understand whether the software is suitable, and how it can be used for particular tasks and conditions of use.*" [5]
- Learnability: "*The capability of the software product to enable the user to learn its application.*" [5]
- Operability: "*The capability of the software product to enable the user to operate and control it.*" [5]
- Installability: "*The capability of the software product to be installed in a specified environment.*" [5]
- Configurability: "*This concern is relevant for all systems having a certain complexity, and where adaptability and flexibility is important.*" [1]

3 Architecture: Context view

The purpose with the context view is to describe all aspects of the Target System's environment, which is of importance to be able to document all the interfaces between the Target System and its environment, and what the Target System is intended to do in its environment.

According to MAFIIA [1] we define the Business aspects model in Figure 1.

3.1 Business aspects model

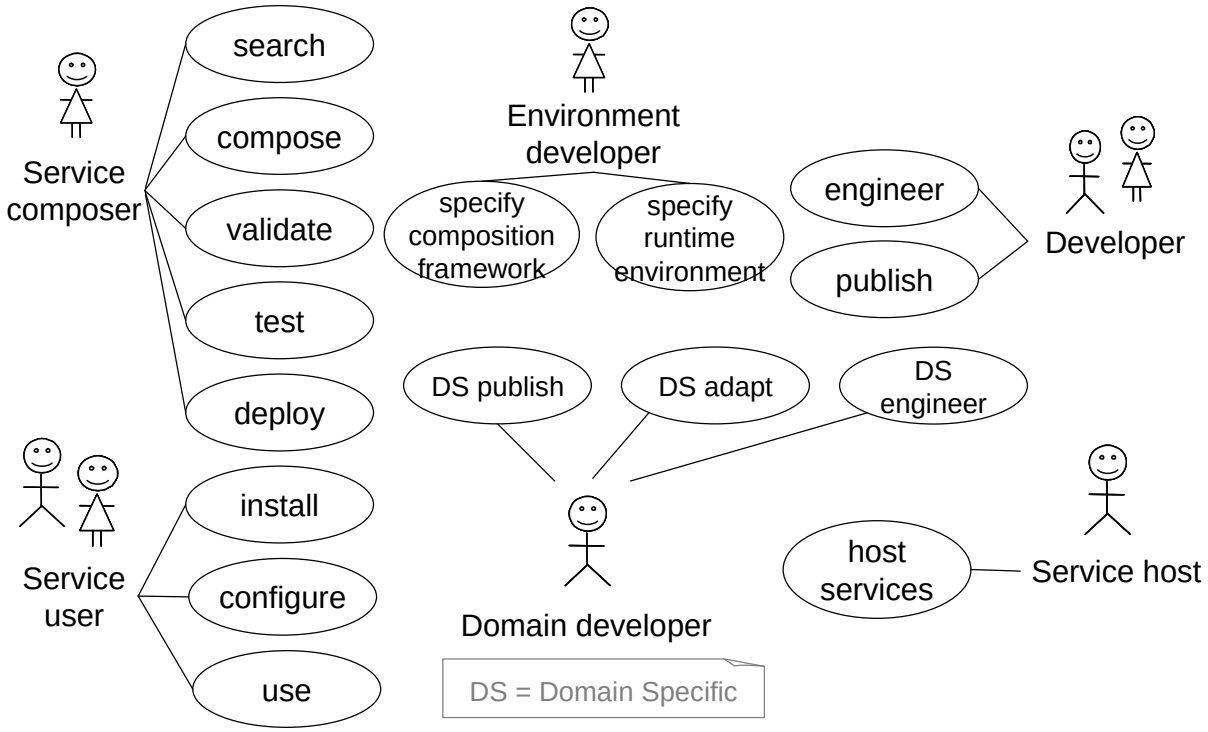

Figure 1: Business aspects model

Note:

1. One and the same person can play both Composer and/or Service user
2. We distinguish between *primary* and *ordinary* Service users, the latter participates without requiring any specific deployment of services or components used by a user service
3. One and the same person can play both Environment developer, Domain developer and/or Developer

We suggest that sharing compositions is not given emphasis at this stage in the project. Many things have to be put in place to support sharing (privacy, security, trust, payment etc).

Table 1: Specification of actors (stakeholder) and activities of the business aspects model

Stakeholder / activity	**Description**
Domain environment developer	**creates composition framework and runtime environment**
specify composition framework	specify composition framework (such as roles, relationships, rules, collaborations, notations, basic components and main tasks with extension points) for a domain
specify runtime environment	specify runtime mechanisms (such as glue logic interpreter,

	deployment, discovery mechanisms) for a domain
Domain developer	**adapts generic solutions to the needs of the domain**
DS engineer (DS: Domain specific)	make new services and components that fulfil needs of the domain, e.g. realize extension points of the DS framework
DS adapt	adapt existing services and components so they fulfil the specific needs of the domain, such as modifying graphical icons and names, and adding domain specific semantics
DS publish	publish services and components that fulfil specific needs of the domain, e.g. realize extension points of the DS framework
Developer	**creates and publishes generic software components**
engineer	create software components that fulfil generic needs
publish	publish software components so they can be found and used
Service host	**hosts domain specific services to end users (e.g. a positioning system in the Wireless Trondheim network)**
host services	host software and services to fulfil domain specific needs of end users, such as Group Management for City Guide
Service composer	**composes and configures services for end users**
search	look for and find services and components among the domain specific ones published that can fulfil needs of the service
compose	create a service composition from domain specific services and components, according to the needs of some end users
test	test a service composition
deploy	deploy the various services and components involved in a service composition to devices and servers
(primary) Service user	**installs, configures and uses software components on servers and devices**
install	install software services & components on servers and devices
configure	configure software services & components on servers and end user devices
use	use services from the user's current device
Ordinary Service user	**takes part in services without specific deployment**
use	use services from the user's current device without requiring any specific deployment of services and components

3.2 Environment systems model

The following systems are found in the environment of the system (called UbiSys for short):

- External services
  - Service APIs used a design time
  - Services provided at runtime
- Software development tools, including:
  - Eclipse (for design tool integration and software development)
  - Android SDK (for software development for certain handheld devices)
  - Arctis (for software component development)
- Runtime support systems, including:
  - Android (supporting certain handheld devices)
  - ActorFrame (component deployment and runtime support on servers and certain clients)

See Figure 2 in next section.

3.3 Business to system mapping model

In Figure 2 we define the border between the system(s) of UbiSys and the environment.

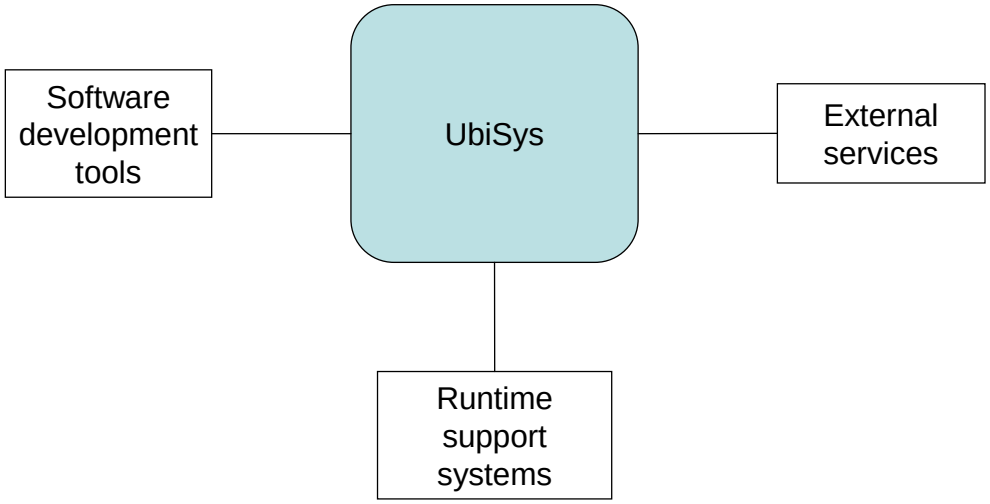

Figure 2: System border (overview)

We consider the system UbiSys to consist of three distinct subsystems:

- A composition system for ICT professionals (domain developers and domain environment developers); we call this *UbiCompPRO*.
- A composition system for non-ICT professionals (composers); we call this *UbiComposer*
- A runtime system for end users (installer and primary end-user); we call this *UbiCompRun*

A more detailed system boundary diagram is shown in Figure 3.

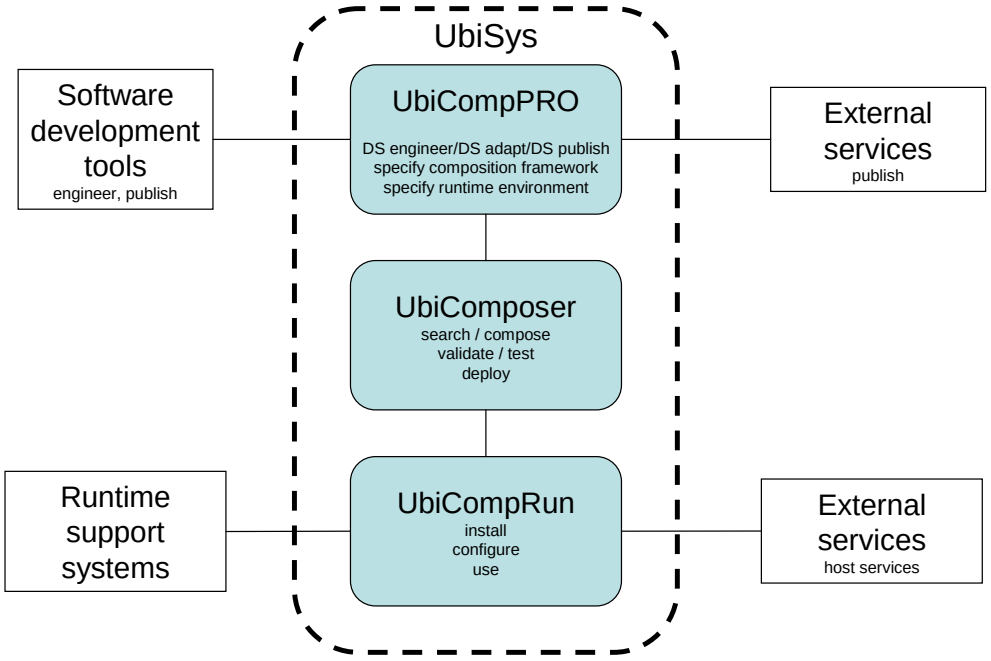

Figure 3: System borders (detailed)

In Figure 3 we have listed the activities of Figure 1 that are supported by the UbiSys subsystems, as well as the activities supported by the external systems.

Note that some activities are somewhat on the borderline, such as *host DS services*, which partly are within UbiSys and partly in the environment; we have depicted them in the former, since they are involved in the design process and must be hosted in the runtime system.

An overview of the flow of internal artefacts is shown in Figure 4.

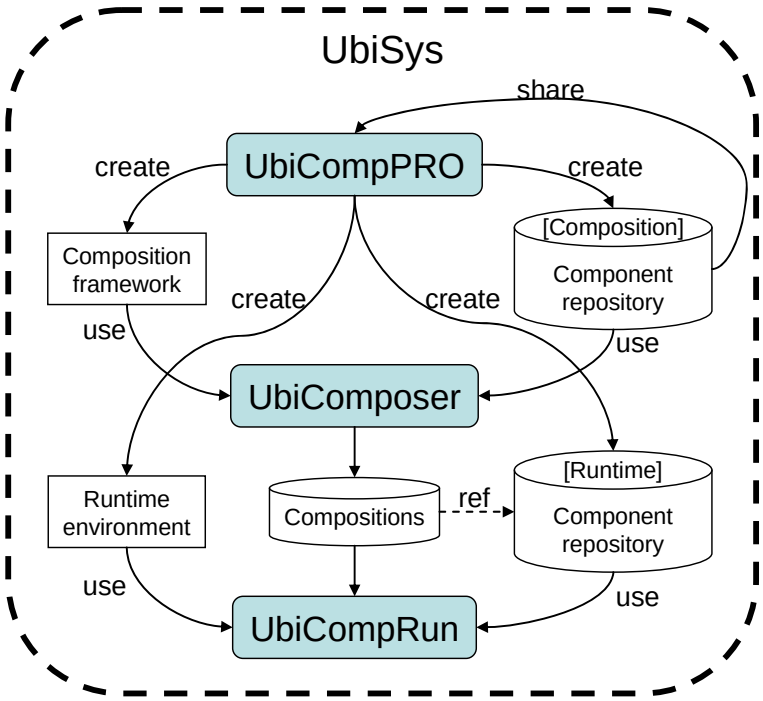

Figure 4: Flow of internal artefacts (overview)

4 Requirement view

The purpose of the requirement view is to document all specific requirements related to the Target System.

4.1 Requirements model

At this stage we make to with a textual description, what MAFIIA calls the Requirement specification form. Model diagrams can be developed later as found necessary.

A list of relevant requirements is given in Table 2 below.

Req. id	Req. description	Acceptance test	Priority (high | medium | low)
G1	Existing solutions to end user service composition should be used as far as possible, such as mashup editors	Inspection	Medium
G2	Generic, existing component and services should be used as far as possible	Inspection	Medium
G3	Generic tools to support Domain Developers and Domain Environment Developers should be used as far as possible	Inspection	Medium
G4	Adaptation to a specific domain should not require much effort (this refers to the activities undertaken by Domain Environment Developer and Domain Developers)	Inspection	Medium
G5	UbiSys should be as domain independent as possible	Inspection	Medium
G6	The UbiCompPRO tool should be domain independent	Inspection	Medium
U1	Understandability: UbiComposer must make it easy for the Composer to understand what software can be composed with it, and how to create a composition which will achieve the users goal	Inspection Evaluate with users?	High
L1	Learnability: UbiComposer must be easy to learn for composers without professional programming background. It should make it easy to get started creating simple compositions, and provide facilities for incrementally acquiring skills needed for more using more advanced functionality.	Inspection Evaluate with users?	High
O1	Operability: UbiComposer must make it possible for the Composer to create and modify compositions with few steps, and provide an interface for this which is forgiving and which assists the composer in creating correct compositions and testing that the composition behaves as expected.	Inspection Evaluate with users?	High

Req. id	Req. description	Acceptance test	Priority (high | medium | low)
I1	Installability: it must be easy to install UbiCompRun.	Inspection Evaluate with users?	High
I2	Installability: it should be easy to install UbiComposer.	Inspection Evaluate with users?	High
I3	Installability: it should be easy to specify the deployment of compositions to multiple devices.	Inspection Evaluate with users?	High
I4	Installability: it should be easy to install compositions on one or several devices.	Inspection Evaluate with users?	High
C1	Configurability: UbiCompPRO should be configurable to support creating composition frameworks for different domains and using different composition facilities	Inspection	Medium
C2	Configurability: UbiComposer must be configurable to support different domains and different composition facilities	Inspection	High
C3	Configurability: UbiComposer should be configurable to combine different domains and composition facilities within one composition framework for the Composer.	Inspection	Medium
C4	Configurability: UbiCompRun should be configurable to support runtime execution of compositions related to different domains and built using different composition facilities.	Inspection	High
C5	Configurability: UbiCompRun should be configurable to combine runtime execution support of compositions related to different domains and built using different composition facilities.	Inspection	Medium
D1	Distribution transparency: Composers should not need to worry about details about protocols when composing and specifying deployment		
F1	Fault handling: error messages should be understandable for composers and end users		
N1	Naming: the vocabulary used in UbiComposer, including commands and component names, should be understandable for composers		

Table 2: Requirement specification form

5 Component view

The purpose of the component view is to identify and document specific physical or logical components. Component descriptions should be purely functional, described by their data, interfaces and functionality. Note that existing or predefined hardware- or software-units can be treated as components and included as components in the component view.

We analyze the tasks and artefacts that are involved, starting with the tasks involving IT-professionals.

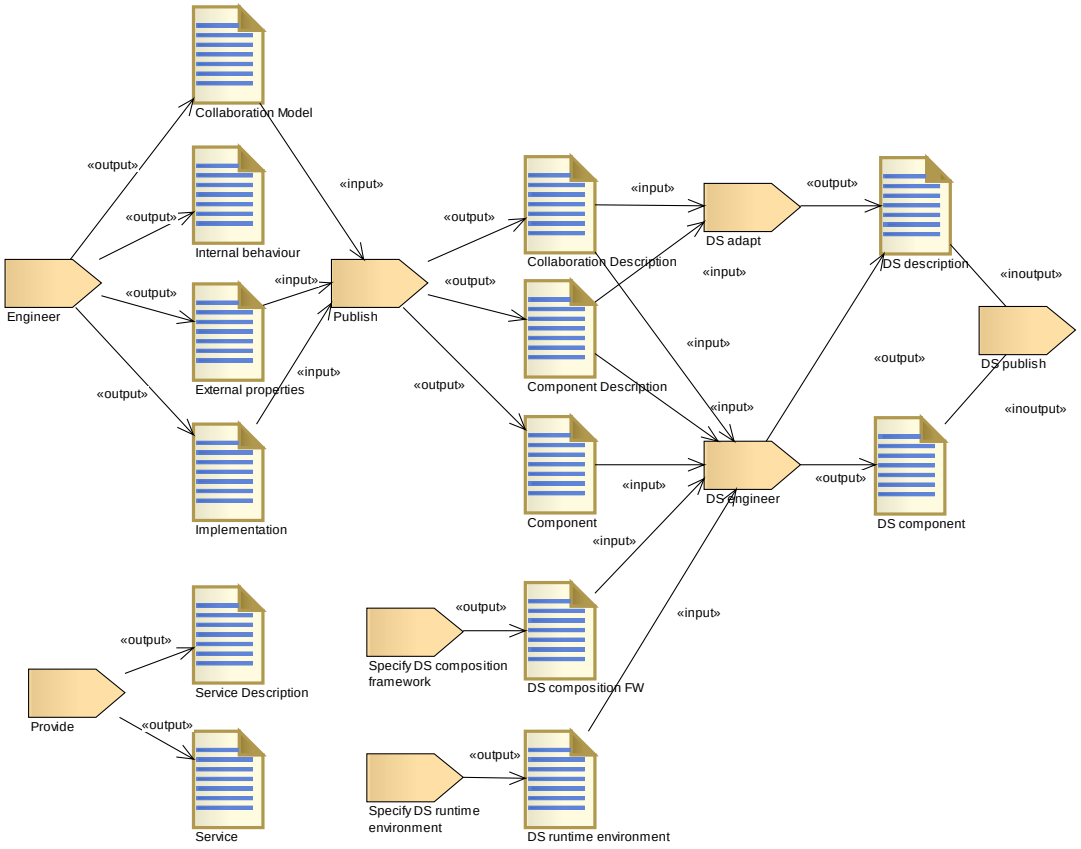

Figure 5: Tasks and artefacts for IT-professionals

In Figure 5 we see the tasks and artefacts involving the roles of Developer, Service host, Domain developer and Domain environment developer, i.e. the IT-professionals.

- The *Developer* performs the *Engineer* and *Publish* tasks. Artefacts involved:
  - collaboration model (e.g. UML2 collaborations): output from *Engineer* and input to *Publish*
  - internal behaviour (e.g. UML state machines): output from *Engineer*
  - external properties (interface specifications): output from *Engineer* / input to *Publish*
  - implementation (programming code): output from *Engineer* / input to *Publish*
  - collaboration description (e.g. goal sequences): output from *Publish*
  - component description (e.g. interface dependencies): output from *Publish*
  - component (programming code, executables): output from *Publish*
- The *Service host* performs the host DS services *(Provide)* task. Artefacts involved:
  - service description (e.g. interface specifications): output from *Provide*
  - service (programming code, executables): output from *Provide*
- The *Domain environment developer* performs *Specify composition framework.* Artefacts:
  - DS composition framework is output

- The *Domain environment developer* performs *Specify runtime environment*. Artefacts:
  - DS runtime environment is output
- The *Domain developer* performs *DS adapt*. It takes the collaboration description and the component description, and produces the DS description
- The *Domain developer* performs *DS engineer*. It takes the collaboration description, the component description, the component (code), the DS composition FW and the DS runtime environment, and produces a DS component and a DS description
- The *Domain developer* performs *DS publish*. It takes the DS component and the DS description as input and revises it (hence “inoutput”).

6 Exemplified architecture using City Guide domain

Below we exemplify the architecture using the application area (domain) called *City Guide*. The goal of City Guide is to make it easy for non-professionals to create city tour services customized to the needs of individual tourists or tourist groups. Such customized services typically provide information about points of interest and navigation support. The tourists use handheld devices to access them.

6.1 Business aspects model

Table 3: Examples of activities of the business aspects model for City Guide

Stakeholder / activity	**Description**
Domain environment developer	**creates composition framework and runtime environment**
specify composition framework *specify composition framework (such as roles, relationships, rules, collaborations, notations, basic components and main tasks with extension points) for a domain*	**roles** (*guide*, *tourist*/*group*, *route*, *place*), **relationships** (*routes* link *places*, *tourists* follow *routes* etc), **rules** (*routes* have a start and a stop *place*), **collaborations** (navigate to next *place* on *route*), **notations** (special icons for roles), basic **components** (*City Guide editor*, group management, navigation support) and main **tasks** (guided tour) with **extension points** (get multimedia info about a *place*)
specify runtime environment: *specify runtime mechanisms (such as glue logic interpreter, deployment, discovery mechanisms) for a domain*	**glue logic interpreter** (specify *City Guide interpreter*), **deployment** (deploy City Guide components to group leader's device using OSGi; configure web server), **discovery mechanisms** (find approved tourist info about *places*)
Domain developer	**adapts generic solutions to the needs of the domain**
DS engineer: *make new services and components that fulfil needs of the domain, e.g. realize extension points of the DS framework*	**make service**: suggest next *place* on *route* **make component**: *City Guide editor*, Group management for City Guide
DS adapt: *adapt existing services and components so they fulfil the specific needs of the domain, such as modifying graphical icons and names, and adding domain specific semantics*	**adapt** *advertisers nearby*: filter out unsolicited advertisers **adapt** *GPS navigator*: select places on *route*
DS publish: *publish services and components that fulfil specific needs of the domain, e.g. realize extension points of the DS framework*	**publish** *City Guide advertisers nearby* **publish** *City Guide GPS navigator*
Developer	**creates and publishes generic software components**
engineer: *create software components that fulfil generic needs*	GPS company: **create** *GPS navigator*
publish: *publish software components so they can be found and used*	GPS company: **publish** *GPS navigator*
External service provider	**provides and publishes generic software services**

provide: *provide software services that fulfil generic needs*	Google: **provide** *advertisers nearby* for Google maps
publish: *publish software services so they can be found and used*	Google: **publish** *advertisers nearby* for Google maps
Service host	**hosts domain specific services to end users (e.g. a positioning system in the Wireless Trondheim network)**
host services: *provide software and services to fulfil specific needs of end users*	Wireless Trondheim: supports publishing and finding points of interest (*places*) in Trondheim
Service composer	**composes and configures services for end users**
search: *look for and find services and components among the domain specific ones published that can fulfil needs*	**find service** *City Guide advertisers nearby* **find component** *City Guide GPS navigator*
compose: *create a service composition from domain specific services and components, according to the needs of some end users*	use *City Guide editor* to **compose** glue logic for a three-hour group tour of central Trondheim for a group of people visiting the town during XP2010
test: *test a service composition*	**test** the composition
deploy: *deploy the various services and components involved in a service composition to devices and servers*	**deploy** of a tour of central Trondheim: *City Guide advertisers nearby* to run on a server hosted by Wireless Trondheim, *City Guide GPS navigator* to run on an Android device
Service user	**installs, configures and uses software components on servers and devices**
install: *install software services and components on servers and devices*	**install** *City Guide GPS navigator* on the Android device of the leader of the XP2010 group
configure: *configure software services and components on servers and end user devices*	**configure** a three-hour tour of central Trondheim on the route web server
use: *use services from the user's current device*	The group leader uses *City Guide GPS navigator* to guide the group through Trondheim, using *City Guide advertisers nearby* as they progress along the route
Ordinary service user	**takes part in services without specific deployment**
use: *use services from the user's current device without requiring any specific deployment of services and components*	Group participants use standard device software (multimedia players etc) to read, listen to and view information about the points of interest (*places*) accessed via the route web service

7 Architecture exemplified with Ambient Assisted Living service

Below we exemplify the architecture using the application *Reminder* from the problem area (domain) of Ambient Assisted Living. The goal of Reminder is to make it easy for *caretakers* (e.g. next of kin) to create reminders, guidance and monitoring services customized to the needs of *subjects of care* (e.g. a slightly dement senior citizen).

The customized services remind the subject of care of upcoming events (for instance a doctor's appointment), where they take place and how to get there (navigation support), and allows the caretaker follow the subject of care and be alerted of a deviation from plan. The subject of care has a reminder touch-screen at home and a handheld device for navigation support and contact with caretaker.

7.1 Business aspects model

Table 4: Examples of activities of the business aspects model for Reminder

Stakeholder / activity	**Description**
Domain environment developer	**creates composition framework and runtime environment**
specify composition framework *specify composition framework (such as roles, relationships, rules, collaborations, notations, basic components and main tasks with extension points) for a domain*	**roles** (*subject of care*, *caretaker*, *event*, *route*, *place*), **relationships** (*routes* link *places*, *subject of care* follows *route* etc), **rules** (*events* are due at a certain *time* at a certain *place*, *routes* have a start and a stop *place*), **collaborations** (navigate to next *place* on *route*, track *subject of care*), **notations** (special icons for roles, events, places), basic **components** (*Reminder editor*, event scheduler, position, navigation support) and main **tasks** (remind of next *event*, navigation along *route* to *place*) with **extension points** (get current travel info about a *route*)
specify runtime environment: *specify runtime mechanisms (such as glue logic interpreter, deployment, discovery mechanisms) for a domain*	**glue logic interpreter** (specify *Reminder interpreter*), **deployment** (deploy components to *subject of care's* mobile device; configure web server and reminder screen), **discovery mechanisms** (find info about *events*)
Domain developer	**adapts generic solutions to the needs of the domain**
DS engineer: *make new services and components that fulfil needs of the domain, e.g. realize extension points of the DS framework*	**make service**: alert *caretaker* of out-of-bounds *position* of *subject* according to current planned *route*. **make component**: *Reminder editor*, Event management for Reminder
DS adapt: *adapt existing services and components so they fulfil the specific needs of the domain, such as modifying graphical icons and names, and adding domain specific semantics*	**adapt** *events nearby*: filter out unsolicited events **adapt** *GPS navigator* to select next *place* on *route* and give spoken navigation assistance to *subject of care*

DS publish: *publish services and components that fulfil specific needs of the domain, e.g. realize extension points of the DS framework*	**publish** *Reminder events nearby* **publish** *Reminder GPS navigator*
Developer	**creates and publishes generic software components**
engineer: *create software components that fulfil generic needs*	GPS company: **create** *GPS navigator*
publish: *publish software components so they can be found and used*	GPS company: **publish** *GPS navigator*
External service provider	**provides and publishes generic software services**
provide: *provide software services that fulfil generic needs*	Google: **provide** *events nearby* for Google maps
publish: *publish software services so they can be found and used*	Google: **publish** *events nearby* for Google maps
Service host	**hosts domain specific services to end users (e.g. a positioning system in the Wireless Trondheim network)**
host services: *provide software and services to fulfil specific needs of end users*	Wireless Trondheim: support publishing and finding *events* of interest to senior citizens at *places* in Trondheim
Service composer	**composes and configures services for end users**
search: *look for and find services and components among the domain specific ones published that can fulfil needs*	**find service** *Reminder events nearby* **find component** *Reminder GPS navigator*
compose: *create a service composition from domain specific services and components, according to the needs of some end users*	use *Reminder editor* to **compose** glue logic for an event involving the *subject of care* taking public transport (a bus) to a *place* in town, being there in time for the *event*, and monitoring of off-track/off-schedule occurrences to *caretaker*.
test: *test a service composition*	**test** the composition
deploy: *deploy the various services and components involved in a service composition to devices and servers*	**deploy** an event reminder: *Reminder events nearby* runs on a server hosted by Wireless Trondheim, *Reminder GPS navigator* on an Android device
Service user	**installs, configures and uses software components on servers and devices**
install: *install software services and components on servers and devices*	**install** *Reminder GPS navigator* on the Android device of the subject of care
configure: *configure software services and components on servers and end user devices*	**configure** the home screen of the subject of care with info about the event.

use: *use services from the user's current device*	The subject of care learns of the event "17:30 today: Meeting for pensioned academics at Trondheim Technical Museum" added by *Reminder events nearby* from the home screen, confirms attendance by touching the screen, and uses *Reminder GPS navigator* to guide herself on and off the right bus in time to get to the meeting.
Ordinary service user	**takes part in services without specific deployment**
use: *use services from the user's current device without requiring any specific deployment of services and components*	Caretaker uses standard web browser etc to view information about events the subject of care is attending, and is notified of any off-track/off-schedule occurrences via email and/or SMS, in which case the caretaker can call the subject of care.

8 References

[1] The MAFIIA handbook – An architectural Description Framework for Information Integration Systems, Report STF90 A05139, SINTEF ICT, February 2003 (ISBN 82-14-03816-2)

[2] Szyperski, C., "Component Software: Beyond Object-Oriented Programming", Addison Wesley, 1997 (2nd ed. 2002, ISBN 0-201-74572-0)

[3] Lieberman. H., Paternò, F., Wulf, V., “End-User Development”, Springer, 2005 (ISBN 1-4020-5309-6)

[4] IEEE Software, Special issue on End-User Software Engineering, vol. 26 (5), 2009

[5] ISO/IEC 9126-1:2001(E)