Winter 12-1-2010

A study on Mobile Requirements Elicitation by Boilerplate Requirements Specification Language

Sundar Gopalakrishnan

Guttorm Sindre

A study on Mobile Requirements Elicitation by Boilerplate Requirements Specification Language

Sundar Gopalakrishnan, Guttorm Sindre, Department of Computer and Information Science, Norwegian University of Science and Technology, Trondheim, Norway.
E-mail: sundar@idi.ntnu.no, guttorm.sindre@idi.ntnu.no

Abstract

With the increasing use of mobile information systems, mobile devices are being used for gradually more complex tasks. Therefore it is also necessary to pay more attention to requirements methods for such systems. One way of supporting requirements engineering is through templates for how to write requirements, often guided by taxonomies. In this paper we propose templates especially for mobility-related requirements, based on a combination of so-called requirements boilerplates as presented by Hull et al. and a taxonomy of mobility-related requirements presented in a previous publication by ourselves. The proposed requirements templates are illustrated by examples, and our work indicates that mobility-related requirements specification may benefit from the use of boilerplates as long as natural language remains an important part of such specifications.

Keywords

Mobility, taxonomy of mobility, Requirements Specification Language (RSL), Boilerplate RSL, requirements template.

1 Introduction

Requirements engineering [1] is an important task in a system development project, finding out what system to build, for instance as part of a contract between customer and developer, and as a basis for design and testing. A number of different techniques have been proposed for the elicitation and specification of requirements, ranging from informal to formal, textual or diagrammatic, and covering a lot of different perspectives, such as object-oriented, process-oriented, rule-based, goal-oriented, actor-oriented, etc. [2,3]. Some techniques are generic, while others are targeted for special system domains (e.g., [4] for e-commerce) or for certain types of requirements (e.g., [5,6] for security requirements).

With the increasing use of mobile information systems, mobile devices are being used for gradually more complex tasks – meaning that challenges with uncertain goals, conflicting requirements, and design trade-offs also become bigger, mandating an increased attention towards good requirements specification. Therefore it is also interesting to investigate requirements techniques specifically for mobile systems - would such systems best be developed by specifically targeted techniques, or by the same techniques as used for requirements engineering in general?

In spite of lots of academic research on formal and semi-formal requirements specification languages, natural language remains - by far - the most usual representation of requirements in the software engineering industry, for instance in the form of "The system shall..." requirements or textual use cases [7], often written in plain word processing or spreadsheet tools. It is therefore important to provide tools that might support the quality improvement of requirements written in natural language, and this would also be an interesting starting point for investigating specific techniques for mobile systems requirements: How to support better quality in mainstream textual requirements for such systems?

In previous work we presented a taxonomy for mobility-related requirements [8]. This taxonomy tries to answer what different kinds of requirements specifically related to mobility would typically come up for a mobile information system, i.e., in addition to other categories of requirements that are well known from before (such as functional requirements, security requirements, etc.). On the other hand, the taxonomy hardly provides any guidelines on exactly how to write these various types of requirements [6,9]. A natural next step is therefore to make some requirements templates based on this taxonomy. A simple and common sense approach to such templates are so-called boilerplates as suggested in [10]. An example of such a boilerplate and its usage could be (quoted from the boilerplates webpage, [11]):

Example boilerplate:

The <user> shall be able to <capability>
at a maximum rate of at least <quantity> times
per <time unit>.

Example instantiation:

<user> = order entry clerk; <capability> = raise an invoice <quantity> = 10; <time unit> = hour

giving

"The order entry clerk shall be able to raise an invoice at a maximum rate of at least 10 times per hour."

(end quote).

The current catalog of boilerplate templates at [11] does not include any category specifically for

mobility-related requirements. The main research questions for the current paper are therefore:
RQ1) Will the previously defined taxonomy for mobility-related requirements lend itself to refinement into boilerplates?
RQ2) Will the resulting boilerplates have any advantages in supporting the elicitation and specification of mobility-related requirements?

The rest of the paper is structured as follows: Section 2 discusses related work. Section 3 summarizes briefly our existing mobility-related requirements taxonomy from previous work. Section 4 provides boilerplate adaptations from repository for mobility-related requirements. Section 5 provides some examples with mobility boilerplates. Section 6 offers discussion and concludes the paper.

2 Related Work

There are several ways to support quality assurance of natural language requirements, by means of guidelines or tools, either (i) to ensure high quality of requirements while writing them, or (ii) after they are written. In both cases, several underlying means may be applied, such as recommended guidelines [12], patterns [7] or templates [13] to indicate how each requirement should be written, or ontologies [14] and taxonomies [12,15,16] for more in-depth recommendations on the semantical content of requirements and what types of requirements to include in a specification, as well as various natural language parsing techniques, either of pre-existing documents to elicit requirements [17] or of the written requirements for the purpose of analysis and validation [18].

The work presented here is naturally closest to the boilerplates approach of [10], attempting a similar style and simplicity of the templates. In particular, it is inspired by some recent work [19], which has taken place in the context of the EU project CESAR [20]. However, CESAR looks at safety requirements for embedded software, while our work focuses on mobility-related requirements in information systems, has investigated adaptations to the boilerplates catalogue to specify safety-related requirements. [19] is also much more ambitious in that it has an underlying ontology and a tool that can also be used for suggesting requirements based on natural language investigations of safety standards, while our work is initially just a simple proposal to support the writing of mobility-related requirements by means of boilerplates, without any underlying ontology [21]. The main novelty of our work, on the other hand, is that it looks specifically at boilerplates for mobility-related requirements, which to our knowledge has not been done in any previous works.

3 Taxonomy of mobility-related requirements

Our existing taxonomy for mobility-related requirements [7,22] is shown in Fig. 1, much inspired by a similar taxonomy by Firesmith [15] for security-related requirements. A central part of both these taxonomies is the combination of achievement levels and challenges, i.e., given some challenge <X> the system should achieve <Y>. For security the challenge might be a certain type of attack (e.g., an intrusion attempt to the system) and the achievement level something the system should be able to do in this case (e.g., detecting or preventing the attack). For mobility the challenge might instead be that the user needs to travel e.g. at high speed, and still the system needs to sustain a network connection, give good navigation advice, etc. In more detail, the proposed taxonomy includes four categories of requirements:

- **mobility requirements**: purely specifying some level of mobility, without indicating design solutions. i.e., specifies mobility challenging factors and mobility achievement levels. Examples for mobility challenge factors may be:
  - the speed of movement needed (the larger the speed, the more difficult it might be to support the movement or provide non-degraded service while moving)
  - the area / range of movement (the larger the area, the more difficult)

  Examples for mobility achievement levels may be:
  - ability to (actively) move. This could be particularly relevant for embedded systems, e.g., where a software application is running an engine and steering system, but less relevant for enterprise information systems, which is our main concern
  - ability to facilitate movement. e.g., real-time positioning, mapping and navigational services
- **mobility-system requirements**: requirements associated with sub-systems whose purpose is to support mobility. It obviously depends on whether the system needs to be in a particular network or not. Examples may be:
  - positioning system
  - network scanning and acquiring system (looking for available networks)
  - service scanning and acquiring system (looking for available services)
- **mobility constraints**: design decisions ensuring mobility that have been lifted to

the requirements level. Examples may be:

- decision to use one specific standard for mobile communication
- decision to use one specific type of mobile equipment, or equipment compatible to that
- decision to use one specific operating system for mobile applications
- decision to use one specific network system for safety/cost reduction

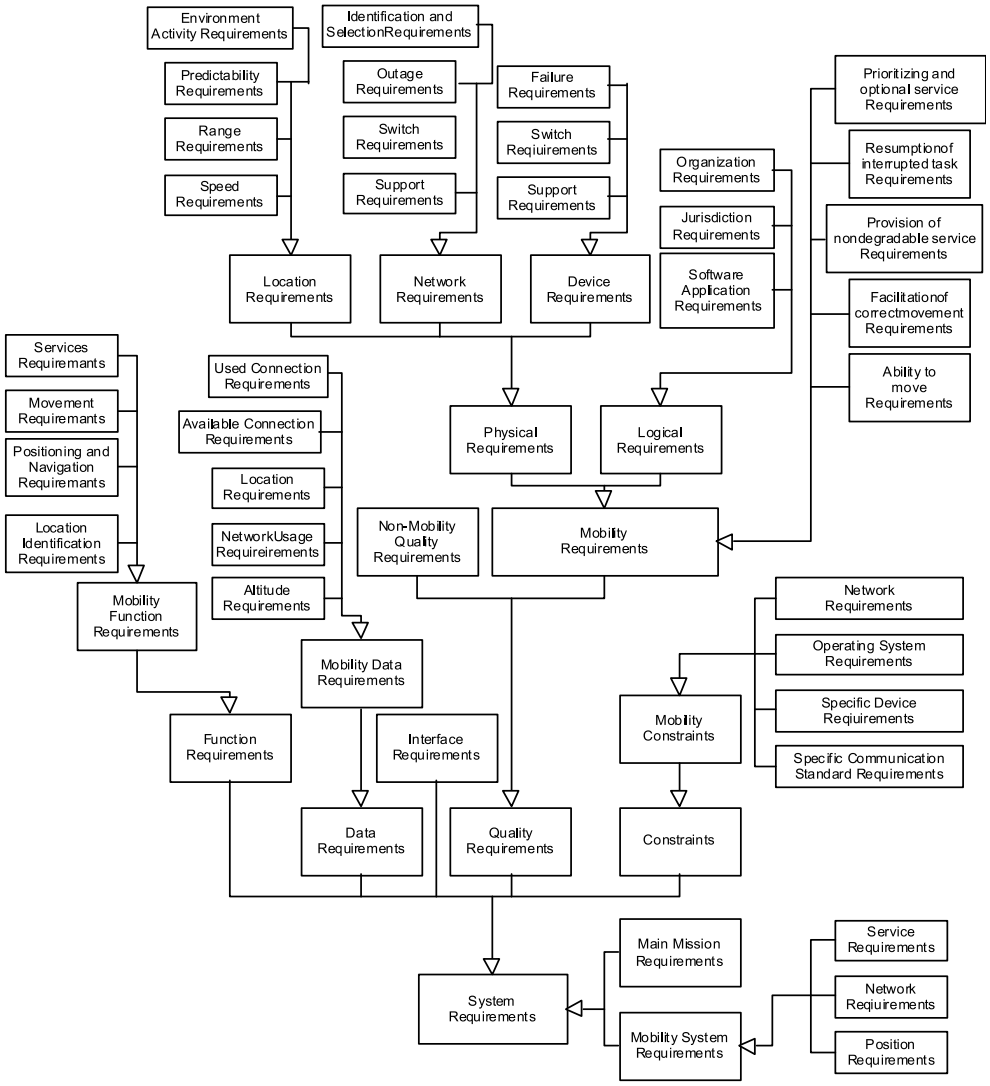

Fig. 1. Taxonomy of Mobility-related Requirements

- **mobility data requirements**: data requirements associated with subsystem which support data and mobility. Examples may be:
  - Limitation with the data network usage
  - Altitude affects limitation of data
  - Available and connections used
  - Location

With this mobility taxonomy briefly described above, we can see how requirements elicitation techniques can be useful to elicit naturally the mobility requirements in the following sections.

We discuss about the three possible techniques among the available techniques and provide initial analysis with related to mobility requirements. This study work analyzes in depth on boilerplate RSL since initial analysis provides the boilerplate RSL having predefined repositories [11] than other RSL techniques overlooked.

4 Refining the taxonomy into boilerplates

In the following four sections we will discuss how the various parts of our taxonomy as presented in section 3 may be refined into boilerplates. To the

extent possible, it would be interesting to use boilerplates as-is, maybe just adding clauses to give more detailed support to requirements authors. But in some cases it might also be necessary to introduce entirely new boilerplates, in case the existing ones do not cover our needs.

4.1 (Pure) Mobility Requirements

For the pure mobility requirements we have to face two questions: what boilerplates to use for the challenges and what boilerplates to use for the achievements? Then, it will hopefully be straightforward to combine these into templates for complete requirement statements.

As for the challenges, the most obvious candidates in the boilerplate catalogue [11] is a BP63 "If <operational condition> and a number of boilerplates with the phrase "while <operational condition>", namely BP36, BP39, BP43, BP44, BP53, and BP65. The main motivation for having so many seemingly similar yet differently numbered boilerplates appears to be that they are subclauses to other boilerplates related to different quality criteria, such as rapidity, timeliness and sustainability, as indicated in the rightmost column in the table of boilerplates appearing when clicking the "VIEW BOILERPLATE REPOSITORY" in [11]. Anyway, the "if / while <operational condition>" phrase can certainly be relevant for mobility requirements too, but as such it gives only limited support to the requirements author, since there are many different kinds of operational conditions that could come into play in mobile information systems. Currently, the author only gets the word "while" for free, the rest of the operational condition will have to be written manually for each requirement without any particular guidance. More help could therefore be achieved if we are able to identify typical operational conditions that could be interesting for mobile information systems and make more detailed Mobility boilerplates (MBP) for these. So, some possible subclauses (or sub-boilerplates) here could be:

1. **speed:** related to the challenge of speed, the operational condition might be specialized to something like
   MBP1: *<way of moving> at a speed of (maximum | minimum) <quantity> <unit>*,
2. where way of moving could again be instantiated to e.g. driving, walking, running, flying a helicopter, riding a bus or train, etc. So, an example requirement could start like "While driving at a speed of maximum 100 kmph..." and then to be completed by some achievement part (here for instance, "...the application shall be able to give updated arrival time estimates every minute.". It could be assumed that the word maximum would be used more often than minimum, since the challenge normally increases with increasing speed, either when it comes to sustaining a network connection, updating information rapidly enough, forecasting the user's position, or whatever.
3. **range:** related to the challenge of range, the operational condition could be specialized to something like
   MBP2: *in <named geographical area> | along <route> | within <range> <measure> from <location point> | within area covered by <network operator>*
- probably there are also other alternatives that could prove interesting when we start looking at a bigger number of example requirements like:
  "*…more than <quantity> <unit>s from <object>"*
4. **predictability:** related to the challenge of predictability, the operational condition could be something like
   MBP3: *moving according to <a prescribed route> | deviating at most <quantity> <unit> from <a prescribed route> | ...*
   Again, there are probably more alternatives to be found when we look at bigger example sets of requirements.
5. **environment:** here, the operational condition could be
   MBP4: *in <terrain type> | indoors | outdoors | in the air | on land | on water |underwater | ... | in environments with <certain condition>*, where <certain condition>
   could again be noisy, crowded, highly trafficked, or something else. Making various templates also for this would probably be to go too far – the user necessarily has to fill in something for himself, too, as we cannot guess in advance all different kinds of requirements that might emerge.
6. **network:** related to the challenges of identification and selection of networks, outage, switching and supporting of the networks could be
   MBP5: *to<identify network type> |3G | EDGE | GPRS | network in <specific network provider> |Telenor | Tele2 | ... |where <network connectivity>*
   could be continuous, high speed connectivity to the network…-the network user to fill whether he needs continuous networking by switching with networks like that.
7. But also some of these requirements could already be satisfied by existing boilerplates, e.g., requirements to be able to resume a task after an outage or to transfer a session to

another device in case of device breakdown might be satisfied by boilerplates BP32 + BP34 as they stand now, e.g. "The user shall be able to resume the session within 1 minute from the network connection returns after a breakdown".

8. **device:** related to the challenges in switching devices could be
   MBP6: *switching from<device type1> |IPHONE | IPAD |...| Notebook* | to < *device type2> |IPHONE | IPAD |...| Notebook* |
   - where the user switching his devices from one to other portable devices

So, for these types of requirements, one could investigate if mobility achievement types could result in more detailed clauses for <capability> in BP32 and for <action> or <entity> in BP2, and challenge types could result in more detailed clauses for <event>, e.g. low power | power loss | return of power | poor network connection | loss of network connection | return of network connection | ...

For the mobility achievement types, there is not necessarily a lot of new boilerplates needed. For instance, "Provision" is just about providing the same service as otherwise, with some mobility challenge. So, it creates clauses related to the mobility challenge, but for the mobility achievement it should be possible to use existing boilerplates. The same with providing degraded service given some mobility challenge; the requirement will normally be phrased in a positive manner (e.g., what services which are still to be provided, with what level of quality) so this will appear like a normal provision requirement except that what is provided is more limited than normally, but this will be apparent by comparison with the normal state requirement and does not have to be mentioned explicitly in the reduced service requirement.

More special are of course the ability to move or the facilitation of movement, for both these there could be more detailed clauses describing exactly what kind of movement is needed.

- **Ability (to move):** related to the ability of achieving movement, the boilerplate could be,
  MBP11: *The <product> shall be able to <move>*,
  e.g., The robot should be able to move. Note that this is different from BP2 "The <system function> should be able to move <entity>", which does not sound very natural here, e.g., the sentence "The robot engine should be able to move the robot" sounds somewhat contrived compared to the simpler sentence above. In addition to just requiring the ability to move, there could also be some conditions for the movement. Notice that this is different from the "while <operational condition>..." clauses discussed earlier, e.g., "The robot should be able to move at a speed of up to 10 meters per second" is entirely different from "While moving at 10 meters per second, the robot should be able to..." Hence, an additional boilerplate could be
  - MBP 12: *The <product> shall be able to <move> <condition for movement>*,
    where <condition for movement> could again be expanded with clauses about speed, range, predictability or environment as discussed earlier, or also other clauses about energy efficiency, avoiding collisions with obstacles or coordinating the movement with other moving entities.
- **Facilitation (of movement)**: This should be able to use BP32 or BP2, e.g. "The navigation application shall be able to tell the driver a proper route". However, the requirements author could get additional support if this was refined into some more detailed alternatives specifically for facilitation of movement, such as:
  - MBP13: The <system function> shall be able to inform <user> about <current | predicted future location>.
    Typically there might then be additional data requirements to explain exactly what information is wanted about various locations; data requirements will be discussed in section 4.4.
  - MBP14: *The <system function> shall be able to provide <user> with <optimal> directions how to <move> to <wanted location>*.
    Here, optimal could mean a lot of different things, for instance the shortest route or quickest route (which are not always the same, depending on speed limits, traffic congestion, etc.), the cheapest route (e.g. if going with public transportation) or simplest route (e.g., minimum number of switches, if there is no direct bus from A to B) - or maybe the safest route, say if some parts of a town are more crime-ridden than others.
  - MBP15*: The <system function> shall be able to warn <user> if deviating more than <quantity> <length unit> from the <optimal route>*.
    This will come into play for instance if the user takes a wrong turn and the system has car navigation functionality. The quantity here would then indicate something about the accuracy of the system. For the user it might be a lot more helpful with a system which gives a warning when the user is only a couple of meters into the wrong alley - when there might still be time to back to the

intersection and take the right turn - than a system which does not discover the mistake until the user has gone some hundred meters. Even more helpful of course a system which would even discover that the car is in the wrong file, so that it could issue the warning before the mistake has happened (E.g., an audio message saying "You need to get over to the right file as soon as possible, your exit point is only a mile ahead")

- MBP16: *The <system function> shall be able to provide the estimated arrival time to <given destination> (updated every <quantity> <time unit>)*

- **Resumption of service** is relevant after e.g. a network outage or device failure. Again, it should be possible to use existing boilerplates like BP2 ("The <system function> shall be able to <action> <entity>") in combination with for instance BP42 ("...within <quantity> <time unit>s from <event>), e.g., inserting "resume" for <action>, "operation" for <entity>, and "the network connection returns after a breakdown" or "the device is rebooted after a shutdown". Similarly, if the concern is not how quickly you can get back in business when the device or network is again working, but how quickly you can work from the time of the breakdown itself (e.g., using something else than the crashed network or device), this could be achieved using BP32 ("The <user> shall be able to <capability>") in combination with BP34 ("...within <quantity> time unit>s from <event>"), e.g. replacing <event> with "device shutdown" or "loss of network connection", and <capability> with e.g. "transferring his working session to another device" or "...another network", possibly also with some extra demands, e.g., "without any loss of data or context" (which is, of course, quite ambitious, but could be a requirement in certain applications where the user's work is urgent and of high value). So, this could be achieved by existing boilerplates, yet again it could be valuable to introduce new subclauses about resuming operation instead of simply the generic <capability> and about network or device failure (or returning availability) instead of <event>, to give better support to the user in having ideas for relevant requirements for mobile information systems. So, some possible boilerplates here could be:

  - MBP17: The <user> shall be able to transfer the following work tasks: <list of work tasks> to <other device> | <other network>...
  - MBP18: The <system function> shall be able to resume the following operations: <list of operations>...
  - MBP19: ...within <quantity> <time unit>s from (warning about low power | warning about poor network | device shutdown | network outage | device reboot | network comeback)
  - MBP20: ...with <max amount> loss of <data> | <context>

Here, the most ambitious requirement would be to fill in "zero" for amount in MBP20, but if this is not achievable (or would be too expensive), one could also go for something less ambitious, e.g., "max 5 minutes of rework due to loss of data".

4.2 Mobility-system requirements

Unlike the pure mobility requirements of section 4.1, which are composed of a mobility challenge and an achievement level, mobility system requirements can be any ordinary kind of requirements (e.g., functional requirements), only that they relate to subsystems which are there for the sake of mobility. Examples could be positioning systems, navigation systems, movement detectors, network scanning and connection components… etc. Thus, it is reasonable to assume that these requirements can be expressed using ordinary pre-existing boilerplates, e.g. *BP2 The <system function> shall be able to <action> <entity>*, which would work fine for requirements such as "The positioning function should be able to determine the device's geographical position" or "The network scanner should be able to find and identify all available networks". It could be possible to make more detailed boilerplates containing various actions and entities that would be particularly relevant for mobile settings, but again it is a question about the feasible level of detail, as it is hard to predict all kinds of requirements that might come up here. Moreover, since these requirements are in a way more "normal" requirements, it makes more sense to stay with the predefined boilerplates until they are somehow proven insufficient.

Also, the specification of WHERE a system capability is needed, which is highly relevant for functional requirements for mobile information systems, will be possible already with the mentioned templates. For instance, using BP53 ("While <operational condition>...") and BP54 ("...the <user> shall be able to <capability>") - or adding our previously mentioned refinements of the operational condition, it would be possible to write requirements like "While at the patient's home, it should be possible for the home-care assistant to record the patient's symptoms in the system".

4.3 Mobility constraints

A constraint is e.g. some design decision which has for some reason been lifted up to the requirements level, for instance that a certain type of mobile device or mobile operating system shall be used, a certain type of user interface (e.g., smartphone touch screen), a certain mode of communication (e.g. audio), etc. Intuitively, the section on Constraints on the boilerplates webpage would be the most appropriate place to look for inspiration, but the boilerplates found here are not necessarily covering our needs. The templates here focus a lot on "shall not" phrases, i.e. constraints in terms of what the user or system shall not be allowed to do, rather than constraints in terms of design decisions made into requirements. Hence, some new boilerplates might be needed here:

- MBP21: *The <system | application> shall run on <type of device or platform>*
- MBP22: *The inter-device communication shall be able to use <some type of network or protocol>*

4.4 Mobility data requirements

Mobility Data requirements are requirements about what data the system should be able to handle, and could either have a semantic style (defining what various concepts are, e.g., "A customer shall be classified as a VIP customer if and only if having made purchases worth at least USD 100,000 per year for the last 5 years and have never been more than 2 weeks late with a payment") or a syntactic form explaining what types of information is needed, with what attributes, e.g., "The system shall handle information about customers, including customer name, billing address, delivery address, phone no., and contact person name." The boilerplate repository of [9] does not seem to contain any templates fitting this need, but it is easy to suggest some. For the syntactic style of data requirement, the following could do:

- MBP31: The <system | application> shall handle information about <entity>: <list of attributes>

This, of course, would be the same whether the system in question is mobile or not. Some particular adaptations for mobile information systems might be to make data requirements dependent on mobile operation conditions:

- MBP32: While <operational condition>... the <system | application> shall handle information about <entity>: <attributes>
- MBP33: While <operational condition> ... the <system | application> shall not handle information about <entity>: <attributes>

For the semantic type of data requirement, a possible boilerplate might be

- MBP34: <Concept1> shall be defined as <concept2> if <condition>

This will be the same in a mobile information system as in other information systems, and we cannot see any particular mobile adaptations that are obviously useful, so we do not discuss it any further.

In Fig. 2, a prototype interface of Boilerplate for mobility-related system is presented. So the boilerplate practitioners can specify the requirement attributes along with stakeholders and associated capability so that the requirement can be elicited.

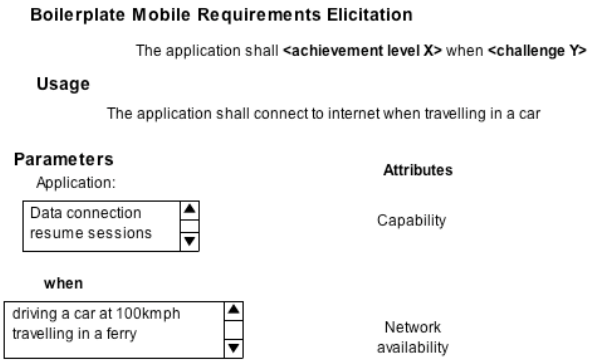

Fig.2 prototype of boilerplate tool

5 Examples

To analyze the boilerplates provided in the previous section, we consider a couple of cases preferably, which are quite different, so that a broader coverage is achieved in terms of testing the boilerplates with different types of requirements. So, there is one case possible where the product itself is supposed to be able to move (e.g., some kind of robot), another where the main goal of the system is to facilitate movement by the user (but by a device which is carried by the user, not moving on its own accord), and finally one case which is not so much about movement as such, but where the user have to perform some information processing tasks while on the move (e.g., the home-care application). We shall see the home-care application as an example than other two since this example more close to IS field and provides easy understanding regarding the scenario. The case is described as below:

The work process within a home care unit, offering practical help and home nursing care to its clients, can be considered as potential case. In the 'Mobile Care'-project [23], it is planned to better

support the mobile aspects of the home care service by providing the employees continuous access to the central health information system (software used in PDA to log/receive info) and other relevant systems from wherever they are using a combined PC/PDA-solution. This is related to the 'Wireless Trondheim'-project [24], which is currently managing and extending a mobile broadband (WLAN) infrastructure for Trondheim. The shift leader distributes patient visits on available personnel in the morning meeting, each homecare assistant then decides on the sequence of visits to be made while still in the office. Then while driving to the patient's home, the assistant prepares for the visit by obtaining some information about the patient (typically through an audio interface, to be less disruptive for concentrating on the driving). Normally, the patient only needs help with day-to-day activities (e.g., shopping, cleaning, taking the right amount of medication), but in case there are some health complications that the assistant cannot handle, a nurse is contacted. Using a system called Gerica accessible by her PDA, the home care assistant can log information about patients on the go. If the health care assistant needs further medical expertise he/she can request help from the nurse at hospital through logging in information via Gerica. The nurses at the hospital get the request and provide further info/advice to be followed by the healthcare assistant (HCA). Finally the HCA finishes her job by reporting at the office.

The mobility-related requirements for this home –care application can be listed as follows:

1. The home-care application in PDA should work on the moving car at 80 KMPH. (MBP1)
2. The HCA shall be able to get briefing information about the next patient while driving within 2 km from the patient's home. (combining BP32 and MBP2)
3. The car navigation system shall be able to automatically warn HQ about delays while HCA is driving in Trondheim municipality.(combining BP32 and MBP2) .(e.g., if the HCA is caught in slow moving traffic and therefore seems to be unable to get to the next patient within a certain time limit)
4. The HCA shall be able to transfer the following work tasks: reporting of delays, recording of patient status, recording of performed patient care, to the backup device within 30 seconds of being warned about low power. (combining MBP17 and MBP19)
5. The patient status recording function shall be able to resume operation within 90 seconds from network comeback, with maximum 2 minutes of rework due to loss of data. (combining MBP18, MBP19, and MBP20)
6. The application shall run on all standard smartphones and PDA platforms with a Norwegian market share of at least 10% per 1.1.2011. (MBP21)
7. The inter-device communication shall be able to use both Telenor's and Netcom's wireless networks. (MBP22)
8. The mobile application shall handle information about clients: name, address, medication needs, dietary requirements, preferred visit times, housekeeping assistance needs, personal hygiene assistance needs, and shopping assistance needs. (MBP31)
9. While driving within the Trondheim municipality the mobile application shall handle information about the road network: street names, intersections, allowed driving directions, speed limits, and any temporary information about blocked or highly congested parts of the roadmap. (combining MBP2 and MBP32)
10. While the mobile device is more than 5 meters from the HCA, it shall not handle information about patients (combining MBP2 and MBP33)

(Here it would of course be assumed that the system at large still handles this information, i.e. probably keeping it on a secure server somewhere, but the mobile device being dislocated from the HCA, one might fear that it has been stolen, lost or forgotten somewhere, and some non-authorized person has it in his/her possession, which makes it a good idea not to be able to access any sensitive info from the device. Note: for this to work, the HCA might for instance have to carry yet another small tracking device on his / her body, and the mobile device would measure its distance to this other tracking device. However, exactly how to achieve this is a matter of design decisions, here we only concern ourselves with the possible requirement).

6 Discussion and Conclusion

Our first research question was whether our taxonomy for mobility-related requirements would lend itself to refinement into boilerplates similar to those in the repository of the boilerplates webpage [11]. Having worked through our taxonomy, we find that mobility-system requirements and mobility constraints can use existing boilerplates, which is not surprising, since these categories of requirements are pretty much like other system requirements and constraints, only that the target system is mobile. For data requirements, we found no boilerplate in the repository, but it is easy to suggest one. This boilerplate, however, contains nothing that would be specific to mobility-related data requirements. So probably, these could use exactly the same boilerplate(s) as any other data requirements. True, in mobile information systems, the user's information need may depend on the

user's location and movements. For instance, with a smartphone-based system for guiding tourists on city walks, you would expect information about certain attractions to become available when the user gets close, e.g., "On your right hand side you will now see the Modern Art Museum", followed up maybe by more detailed information on opening hours, ticket prices, special exhibitions that are on right now, etc. However, data requirements concern themselves with what data the system should be able to handle, not so much with how these data are acquired or how and when they will be presented, which would be more dependent on functional requirements and maybe usability requirements. In this respect, it is reasonable to assume that mobility-related data requirements should be able to use the same boilerplate as data requirements for non-mobile systems.

Our main effort with new or adapted boilerplates has therefore been with the so-called "pure" mobility-requirements, which combine a mobility challenge and a mobility achievement level. For the mobility challenge, we found that this could use the "While <operational condition>" clause already suggested in the boilerplates library, but that the user could be given increased support if we introduced a number of subclauses detailing typical operational conditions that apply to mobile systems, as indicated by the mobility challenge part of our taxonomy. A fair question is of course whether there is added value in introducing these subclauses, instead of going with the already available "While <operational condition>..." clause, since a common advice is to keep things simple. We believe that there could be a number of advantages with the additional clauses:

- reduced writing time for requirements. The pre-existing clause gives only one word for free ("while") and the rest has to be filled in the by user manually. With a boilerplates based requirements tool offering a drop down menu of subclauses to select from, the user could get more words for free, and only have to fill in a couple here and there. Still, this is only a minor advantage; for a quick typer, the time spent writing even a 50-letter operational condition might not be much longer than selecting a template from a menu and then filling in 5 letters in a couple of places. And anyway, the main usage of time in requirements specification would seldom be the time to type the requirements - the difficulty rather lies in (i) finding the right requirements, and (ii) formulating them in a precise and understandable manner. Hence, our next two benefits are much more important:
- support for finding requirements. A drop down menu of templates would not only work for selection once you know what requirements to include, but could also work as a repository for ideas (e.g. to feed into a requirements brainstorming session for a mass market product) or as checklist (e.g., of questions to ask the user / customer in an interview or requirements workshop for a bespoke product). For instance, seeing that you have in the drop down menu templates concerning the speed, environment and range of movement, you know that you could ask your customer questions like: "Do you have any requirements concerning the speed / environment / range of movement?" Of course, the customer might provide the same answers given a more general question like "What are the operational conditions for this application?", but chances are high that the customer will forget something which might have been remembered with the more detailed list of questions prompted by more detailed boilerplates.
- support for writing requirements in a standardized way. Especially in projects where a lot of people are collaborating, it may be a problem that a lot of different writing styles will be used for the requirements, and various contributors to the requirements specification use language differently, for instance using different terms for the same concepts (synonyms) or the same term for different concepts (homonyms). This makes it more difficult to compare them, check for completeness, overlaps, and inconsistencies. Boilerplates help reducing this problem, but even with boilerplates, the words to be filled in manually are still open for different styles by different requirements authors. So, the more detailed the boilerplate, with fewer words to fill in manually by the requirements authors, the smaller the problem with different individual usages of natural language. Hence, the more detailed boilerplates can help writing more precise and uniform requirements within a large project, and also save the requirements engineers time in wondering how to formulate each requirement.

A typical counter-argument could be: What if the proposed boilerplates are too constraining? What if

a mobile IS project comes up with requirements that do not fit any of the new boilerplates we have proposed, for instance concerning the <operational condition>? Then, maybe a tool which insists on using boilerplates will actually have given the requirements engineers a harder time, rather than helping them? However, the simple answer to this is that if none of our new templates fit, it should of course still be possible for the requirements author to go with the more general "While <operational condition>..." and simply type in the operational condition himself. Hence, the analyst will get help from the new boilerplates when these are relevant, but can resort to simpler and more general boilerplates if they are not relevant. This answer the second research question, whether it is reasonable to assume some advantage from the new boilerplates. It must be noted, however, that so far our approach has only been tried out on some small desktop examples, and more substantial empirical investigations are needed to make a sound claim that there is really a benefit gained from this. Hence, some obvious suggestions for further work are as follows:

- Collecting all boilerplates (existing ones and new/adapted ones) in one systematic catalogue
- Implementing tool support for our boilerplates, and then trying out this tool in e.g. student experiments and industrial case studies.

To achieve a prototype rapidly, it is probably best to build it on top of an existing tool rather than developing something from scratch. One possibility would be to build it on top of DOORS, since the boilerplate originators have already implemented their approach with this tool. Another possibility, which is maybe even easier to achieve quickly, would be simply to implement it as macros in Excel or a similar spreadsheet tool. This may sound a little weird since spreadsheets are not dedicated requirements tools, but actually spreadsheets are used a lot in industry practice for documenting requirements, so a spreadsheet implementation of boilerplates might have good chances for industrial take-up, since companies would then not have to deviate much from their current practice to try out the tool in their projects.

References

[1] Kotonya, G. and Sommerville, I.: Requirements Engineering - Processes and Techniques, Wiley, 1998.

[2] Van Lamsweerde, A.: Requirements Engineering in the Year 00: A Research Perspective, In: Proc. ICSE'00, IEEE.

[3] Nuseibeh, B. and Easterbrook, S.: Requirements Engineering: A Roadmap, In: Proc. ICSE'00, IEEE.

[4] Gordijn, J. and Akkermans, H.: Value Based Requirements Engineering: Exploring Innovative E-Commerce Ideas, *Requirements Engineering* 8(2): 114-134, 2003.

[5] Sindre, G. and Opdahl, A.L.: Eliciting Security Requirements with Misuse Cases, *Requirements Engineering* 10(1): 34-44, 2005.

[6] Pavlovski, C.J. and J. Zou, Non-functional requirements in business process modeling, in Proceedings of the fifth on Asia-Pacific conference on conceptual modelling - Volume 79. 2008, Australian Computer Society, Inc.: Wollongong, NSW, Australia.

[7] Denger, C., Berry, D.M., and Kamsties, E.: Higher quality requirements specifications through natural language patterns. In: Proc. International Conference on Software: Science, Technology and Engineering, Herzlia, Israel, 4-5 Nov. 2003.

[8] Gopalakrishnan, S. and Sindre, G., A Revised Taxonomy of Mobility-Related Requirements, In Proc. International Workshop on Management of Emerging Networks and Services (MENS'09), St.Petersburg, Russia, 12-14 Oct, 2009

[9] Ohnishi, A. Software requirements specification database based on requirements frame model, in Requirements Engineering, 1996., Proceedings of the Second International Conference on. 1996.

[10] Hull, E.C.,, Jackson, K, and Dick, J. Requirements Engineering, 2nd ed., London: Springer Verlag, 2004.

[11] Dick, J. http://www.requirements engineering.info/ (website supplementing [8]), last updated April 2006, last accessed: 12 Oct 2010.

[12] Improving the quality of use case descriptions: empirical assessment of writing guidelines. Phalp, K.T., Vincent, J., and Cox, K. *Software Quality Journal* 15(4):383-399, 2007.

[13] Robertson, J. and Robertson, S.: VOLERE: Requirements Specification Template, Technical Report Ed. 6.1, Atlantic Systems Guild, 2007.

[14] Kaiya, H. and Saeki, M. Using domain ontology as domain knowledge for requirements elicitation. In: Proc. 14th International Conference on Requirements Engineering (RE'06), IEEE, Minneapolis, USA, 11-15 Sep, 2006.

[15] Firesmith, D.G. Common Concepts Underlying safety, security, and survivability engineering. Technical report CMU/SEI-2003-TN-033, Carnegie Mellon Software Engineering Institute, 2003.
[16] Firesmith, D.G., A taxonomy of security-related requirements. In: Proc. International Workshop on High Assurance Systems (RHAS'05), Paris, France, 29-30 Aug, 2005.
[17] Goldin, L. and Berry, D.M.: AbstFinder, a prototype natural language text abstraction finder for use in requirements elicitation, *Automated Software Engineering* 4(4):375-412, 1997.
[18] Gervasi, V. and Zowghi, D. Reasoning About Inconsistencies in Natural Language Requirements. *ACM Transactions on Software Engineering and Methodology* 14(3):277-330, 2005.
[19] Reichenbach, F., Stålhane, T., and Omoronyia, I. Ontology-guided requirements and safety analysis. In: Proc. 6th International Conference on Safety of Industrial Automated Systems (SIAS 2010), Tampere, Finland, 14-15 June 2010.
[20] URL: www.cesarproject.eu, accessed on 10th Oct 2010.
[21] Veijalainen, J., Developing Mobile Ontologies; who, why, where, and how? IEEE, 2007.
[22] Gopalakrishnan, S. and Sindre, G., Taxonomy of Mobility-Related Requirements, In Proc. Int'l conference on Interoperability for Enterprise Software Applications (I-ESA'09), Beijing, China, 20-22 Apr, 2009.
[23] URL: http://research.idi.ntnu.no/trimaks, accessed on 10th Oct 2010.
[24] URL: http://tradlosetrondheim.no/, accessed on 10th Oct 2010.