

Software specifications for developing composable Mobile Learning systems

Magnus Persson

Abstract

A substantial amount of time and efforts in the initial stages of creating a software system is spent capturing requirements, deciding what software to use and creating technical prototypes to gain confidence in the decisions that have been made are correct. This thesis targets the domain of Mobile Learning with the aim of optimizing these initial stages of a new project by identifying commonalities and architectural patterns found in six existing software systems related to research projects in the domain.

The results present a set of requirements, guidelines and/or an initial conceptual architecture that can be extended or adapted to a broad range of software systems.

Keywords

Adaptability, mobile learning, reusability, software engineering.

Table of Contents

1. INTRODUCTION	1
1.1. Purpose and goal	1
1.2. Thesis outline and disposition	2
1.3. Scope and limitations	4
2. PROBLEM DOMAIN SURVEY	5
2.1. Field studies supported by mobile technologies: an overview	5
2.2. Literature study summary	8
3. FEATURE DECOMPOSITION AND ABSTRACTION	10
3.1. Abstracted features	10
3.2. Connecting with architectural patterns	11
3.2.1. N-tiered architecture	11
3.2.2. Event-driven architecture	12
3.3. Summary	13
4. VALIDATION	15
4.1. Requirements engineering	15
4.1.1. Conceptual system architectures	16
4.2. Implementation and testing	18
4.2.1. Framework data model and rationale	19
4.2.2. Relevant terminology	20
4.2.3. Implementation details	21
4.2.4. Unit test results	21
4.3. Case studies	22
4.3.1. Framework adaptability aspect	22
4.3.2. Framework extensibility aspect	24
5. CONCLUSION	26
5.1. Results	26
5.2. Future work	26
APPENDIX A : SYSTEM REQUIREMENTS	30
APPENDIX B : FRAMEWORK DATA MODEL, CLASS DIAGRAM	34
APPENDIX C : DATA MODEL SOURCE CODE	35
APPENDIX D : UNIT TEST EXECUTION	38
APPENDIX E : UNIT TEST SOURCE CODE	39
APPENDIX F : POSTGRESQL TRIGGER GENERATOR	47
APPENDIX G : DATAMATRIXWORKER.PY	49
APPENDIX H : DATA MATRIX IMAGE WITH SOURCE IMAGE	51

Table of figures

FIGURE 1.1. GOALS	2
FIGURE 1.2. PROCESS	3
FIGURE 1.3. PROBLEM DOMAIN SURVEY	3
FIGURE 1.4. FEATURE DECOMPOSITION AND ABSTRACTION	4
FIGURE 1.5. VALIDATION	4
FIGURE 2.1. IMAGE CAPTURE WITH DATA MATRIX ASSOCIATION (YEH ET AL., 2006)	6
FIGURE 2.2. HYCON SERVICE FRAMEWORK ARCHITECTURE (HANSEN AND BOUVIN, 2009)	7
FIGURE 3.1. A N-TIERED ARCHITECTURAL PATTERN	12
FIGURE 3.2. AN EVENT-DRIVEN ARCHITECTURE	12
FIGURE 3.3. FEATURE MATRIX	13
FIGURE 3.4. ARCHITECTURE MATRIX	13
FIGURE 4.1. MoSCoW PRIORITISATION	15
FIGURE 4.2. CONTENT REPOSITORY	16
FIGURE 4.3. SENSOR NET PROXY	17
FIGURE 4.4. LOCATION AWARE ACTIVITY SYSTEM	18
FIGURE 4.5. EVENT AWARE CONTENT AGGREGATION	18
FIGURE 4.6. FRAMEWORK ANALYSIS DIAGRAM	19
FIGURE 4.7. LETS GO SYSTEM CONCEPT	23
FIGURE 4.8. LETS GO CONTENT BROWSER	23
FIGURE 4.9. GeM SUB-ACTIVITY CONCEPT	24
FIGURE 4.10. GeM SENSOR NET PROXY	25
FIGURE 4.11. EVENT SCHEMA	25

1. Introduction

The new generation of mobile and web technologies are changing the way in which we can learn, communicate and collaborate. With regard to software applications, the broad adoption of cheap mobile devices and sensors has led to the development of software systems within the field of mobile learning that can assist and enrich the learning process (Druin, 2009, Spikol and Milrad, 2008) where the research can be defined as:

Research into mobile learning is the study of how the mobility of the learners augmented by personal and public technology can contribute to the process of gaining new knowledge, skills and experiences. (Sharples et al., 2008)

Often, these software systems start out with the idealization that they should be generic in nature so that components of the system can be reused or integrated into another system at a later point. In reality, it can be said that this idealization rarely holds. As the requirements of the system evolve with the participation of stakeholders and other interested parties, it becomes less generic and more tied to the specific purposes of the system. If at a later time, a project entails that an existing system should be reused against another set of similar requirements, the cost and efforts to do this could be insurmountable. In an ideal situation, the existing software system should be forked, relying on previous results and adapting for the new requirements.

Frohberg and colleagues critically analyze 102 mobile learning projects (Frohberg et al., 2009), indicating an abundance of software systems in this area. At review, it can be said that many of these systems should share common components or key aspects that can be abstracted and provided as a framework, giving a foundation upon which project-specific development can be made. This thesis puts a special focus on studying the development of software systems used to support learning activities and field observations with mobile devices. This latest statement serves to define the particular problem to be explored in this thesis that can be formulated as follows: *How can we optimize the software development process of the type of systems defined above by pre-emptively identifying their key common components?*

The notion of optimization in the context of this work should be seen as the process that helps to shorten the development time to the delivery of a finalized system. Therefore it could also be read as *optimization by reusing* where we observe commonalities (such as in architectural design or implementation aspects) over a broad spectrum and attempt to extract these as components which then can be considered for reuse already at an early stage of the software development cycle.

1.1. Purpose and goal

It can generally be said that the cost of a software system is directly related to the efforts spent developing and maintaining its components. This thesis aims to explore the different software components within this specific problem domain with the purpose of identifying commonalities. By surfacing these commonalities, we can cultivate them into a set of requirements that can be introduced at an early stage of any project lifecycle. A secondary goal of this thesis is to provide an implementation utilizing the set or subset of requirements, for the purpose of providing confidence that they are applicable to software systems in the problem area.

The iterative development cycle (Jacobson et al., 1999) slices the development of a software system into four phases: *Inception*, *Elaboration*, *Construction* and *Transition*.

Each phase can have any number of iterations where the results of the last iteration are used as input to the next.

The *Inception* phase starts a project, measuring if it is feasible and constructing test prototypes to validate technical decisions. In this phase we also find high-level requirements engineering to scope the project. In the *Elaboration* phase an agreement has been met to proceed with the project and focuses on refining the requirements and overall system architecture. The result of this phase is an executable system that is considered to be stable enough to be built upon in the upcoming *Construction* phase.

The final *Transition* phase is where the finalized system is adapted and delivered to the end users environment as product, proceeding after defects discovered during beta testing have been fixed and that an agreement is met between the developers and stakeholders that the system is working as intended (Arlow & Neustadt, 2005).

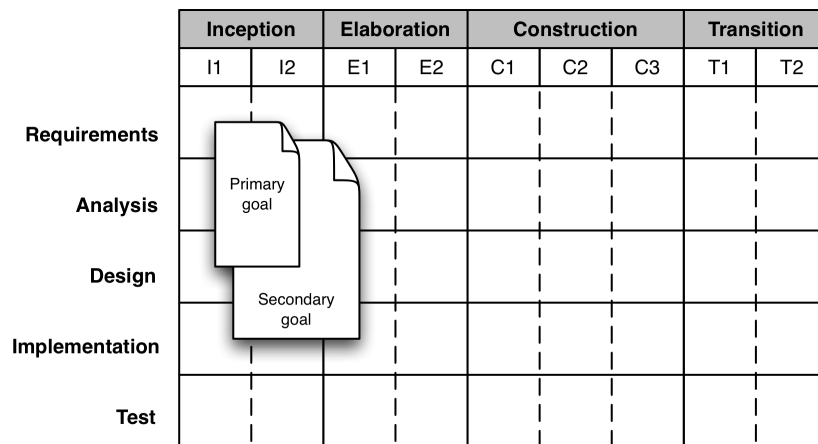


Figure 1.1. Goals

Figure 1.1 illustrates a generic project outline consisting of four phases with five core workflows (requirements, analysis and so on) divided into a number of iterations (I1, I2, E1 and so on). From this figure, we find the first goal of the thesis targeting the *Inception* phase and the secondary goal expanding into the *Elaboration* phase.

1.2. Thesis outline and disposition

The disposition of this thesis in lines the ideas and processes presented in Figure 1.2 below. These processes can be enumerated as follows; *problem domain survey*, *feature decomposition and abstraction* and *validation*. This thesis is structured as follows; chapter 2 presents the results of a literature review focusing on a number of software systems used in the field of mobile learning. This chapter ends with a summary of the main findings that provide the basis for the rest of the thesis. Chapter 3 proceeds by presenting some results connected to feature extraction in connection to the projects presented in the chapter. Chapter 4 presents the requirements of the proposed framework that will then be implemented and validated against test scenarios. The framework will also be used in case studies, applying it to existing projects and discussing extensibility and adaptability aspects. The thesis concludes in chapter 5 providing a discussion over the results and possible directions of future work.

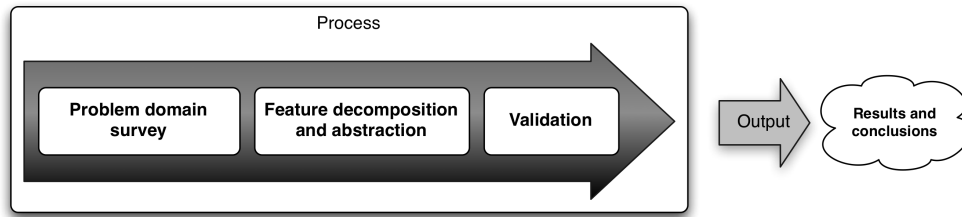


Figure 1.2. Process

Prior to initiating this thesis, an assumption was made that it is difficult to reuse software systems in this specific problem domain as requirements evolve. The grounds of this assumption originate from observing the progression of research projects at the Center for Learning and Knowledge Technologies (CeLeKT) at Växjö University, in particular from a software engineering perspective. What was observed indicated that while the projects were diverse in nature, they did include similar aspects with regard to software design and implementation. Based on this observation, an idea surfaced to identify the required set of features for a software system that could easily adapt to this environment, and present it as a reusable and/or adaptable framework.

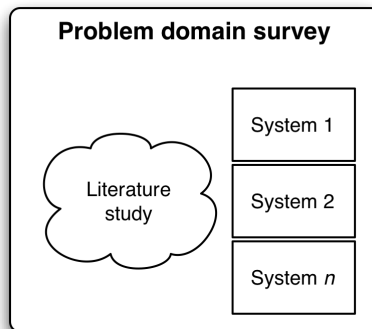


Figure 1.3. Problem domain survey

In order to get a general understanding of these features a study of projects related to the problem domain must first be conducted and is part of the problem domain survey illustrated by Figure 1.3. The projects included in the study were either discovered by searching at academic databases such as Google Scholar or have been referred to me from colleagues at CeLeKT or by my supervisor. Keywords used when searching in the databases have been combinations of the following terms: *mobile learning*, *fieldwork*, *tools* and *visualization*. The criteria applied when selecting candidates is adapted from (Frohberg et al., 2009) and are the following:

- A prototype must exist and be usable.
- The system should be used in an authentic user test with the clear goal of learning something.

In addition, the literature study must to some extent also include research efforts taken in a direction of the problem this thesis presents. The findings here can be used as comparison and grounds for discussion to validate or invalidate the solution.

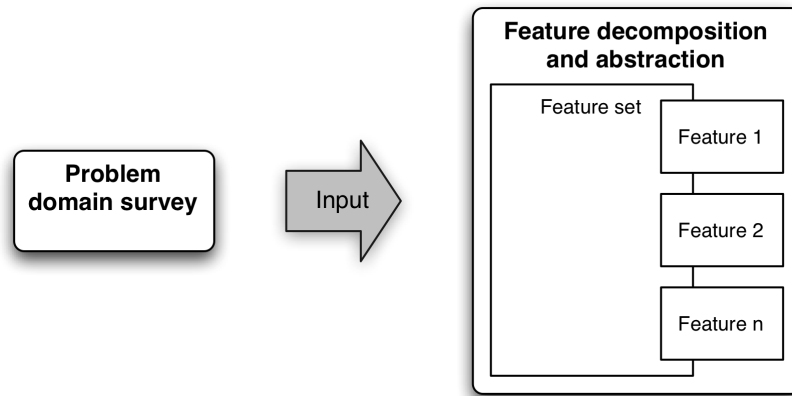


Figure 1.4. Feature decomposition and abstraction

Reflecting upon the results of the study, illustrated as input to the *feature decomposition and abstraction* part in Figure 1.4, will allow us to describe necessary features in abstract terms. Additionally, I identify architectural patterns that together with the abstractions will lay the foundation of what should be included in the framework and how it should eventually be composed.

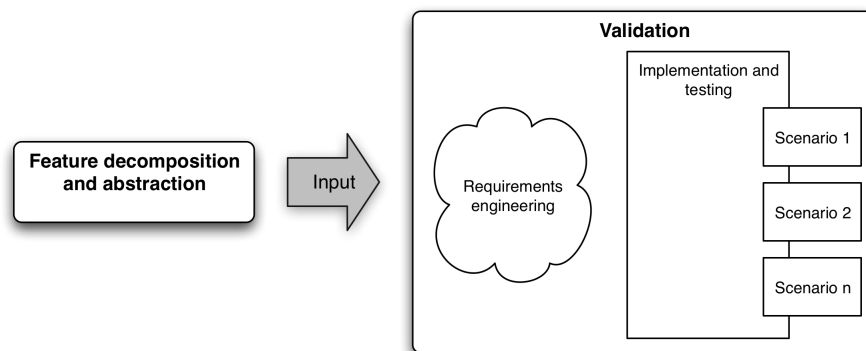


Figure 1.5. Validation

The outcome of the previous part will allow formal requirements to be captured and formalized in the final *validation* part of this process, as illustrated in Figure 1.5. To substantiate that the requirements are valid they will be used and implemented against test scenarios that mimic the candidate projects included in the literature study. As these are sufficiently expressive in nature, they can be implemented using a software-testing framework.

1.3. Scope and limitations

The projects selected for the literature study and feature extraction should not be considered complete but diverse enough to give a decent initial set of features. It must be taken under consideration that this thesis aims to include a framework implementation that can be tested and validated. A larger set of projects would undoubtedly lead to a more feature rich framework but time constraints could lead to an inability to validate the findings. Also, the projects included in the study most certainly will contain components that fall outside the definition of the framework this thesis aims to present. Primarily this means that the work presented in this thesis excludes client applications but not necessarily how they can be used with the framework as a whole.

2. Problem domain survey

This section begins by presenting a literature study encompassing several research projects from the field of mobile learning. More specifically, this study will focus on the technical aspects and based on the findings from the studied projects, an abstracted set of features is formulated and will later be presented in chapter 3. At the end of this chapter, those projects that have been studied will be summarised and connected with architectural patterns that can later be referred to when constructing conceptual software architectures for the framework this thesis aims to present.

2.1. Field studies supported by mobile technologies: an overview

The *Ambient Wood* project pioneered the notion of technology augmented field trips (Rogers et al., 2004); Druin, 2009). In *Ambient Wood*, children visited a woodland area and were equipped with a number of devices to allow the augmented experience. They used walkie-talkies to communicate with each other and also with a remote facilitator that assisted with guidance during the activities. A light and moisture probe allowed the children to record samples while at the same time visualising the readings. Limited range (10 meters) FM radio transmitters were located at different areas of interest in the wooded area, broadcasting uniquely identifiable signals. A small handheld computer extended with a radio receiver that the children carried along with them served as a presentation tool. As the receiver entered an area of interest, the signal was identified and communicated (over wireless networking) to a central server, triggering an event. Depending on the area of interest, various types of digital content was relayed to the children ranging from the playback of pre-recorded audio (via wireless speakers) to the display of images appropriate to the area of interest, such as plants or animals.

PhotoCompas (Naaman et al., 2004a) is a software system that helps users in finding the correct image from their personal photo albums. The system assists by the automatic grouping of images into hierarchies of location and time-based events. These events are proven (Naaman et al., 2004a) to help users more easily locate an image based on their recollected memories of where and when it was captured. By using location and time attributes the system adapts to discrepancies when the image was captured. In the article the authors give an example where when travelling users forget to adjust for time zone differences and thus, given a 12 hour time zone difference, an image captured in the afternoon will actually show as being captured at night. By using a geographic dataset of time zones the system can adjust for the error and give the correct local time when the image was captured. Another automatic categorization done by the system is “Light status”. Here the system utilizes an external data source together with location metadata from the image to give sunrise and sunset information. This is then used to group images as taken during day, dusk, night and dawn. Location metadata is further used in the *PhotoCompas* system to give textual representations of where the image was captured. By querying against a geographical dataset the image can be placed in the country, province/state, county and, if available in the dataset, also the city and park where it was captured. The images are additionally placed into geographic clusters (Naaman et al., 2004b) that give more meaning to the textual representation of the location.

ZoneTag (Ahern et al., 2006) is a software system that involves semi-automatic tag suggestions during the post-capture phase of digital images on mobile devices. At the time of publication, the *ZoneTag* client was being used by 250 users (of which 100 were active users) distributed over 18 countries. The client being employed runs as a background process on the mobile device and keeps track of the users’ current location

based on cell tower ID or GPS data if that is available. Tags are pre-fetched to the client at application start up and later updated if the location changes (by roaming to another cell tower) or every 10 minutes. Once an image has been captured the user can, via a graphical user interface, select suggested tags ordered by importance related to the current context. In the article, a comparison between manual tagging and using suggested tags via *ZoneTag* is made. The results of this comparison show that the amount of tags on an image notably increases when the user is provided tag suggestions at point of capture.

ButterflyNet (Yeh et al., 2006) is a software system that aims to assist field biologists by utilizing notebook-centric interaction. While in the field, the biologists are equipped with a digital pen that digitizes each stroke. The recorded data can later be transferred to a computer saving time otherwise spent transcribing handwritten notes. The paper used with the pen has a dot pattern that enriches the recorded data with when, where and also which page, the stroke was made. To further support biologists in the field, they are also equipped with a prototype “smart camera” where by using a stylus, pen annotations can be made after capturing an image. The camera is also wirelessly connected with the digital pen, a coupling that allows the user to visually mark an area on the paper and associate annotations with the captured image. The article authors refer to this technique as “hotspot association”. Traditionally when collecting specimens in the field, biologists place it in an envelope and tag it with some sort of identification. *ButterflyNet* extends this process by letting the user first photograph the specimen together with visual marker known as a *data matrix symbol*. Since the marker can be machine interpreted an association is established as the photograph is taken. Subsequent images that include the marker carry the association together with any annotations made with the digital pen due to the coupling previously mentioned. Figure 2.1 illustrates different stages of the process in a real environment.



Figure 2.1. Image capture with data matrix association (Yeh et al., 2006)

In the post-activity, the biologist can use the *ButterflyNet* browser application to visualize the material that was recorded while in the field. A timeline component allows the user can navigate the material. The timeline also includes a bar representation of the amount of material captured in the timeframe. The user can also physically navigate the material by using the digital pen and tapping the page of interest, whereby the browser shows the digital version. This technique is also used to recall an image associated with a hotspot. Yeh and colleagues (2006) mentioned in their efforts for future work that they would like more types of digital media (video and audio) and also location as a contextual attribute, which *ButterflyNet* at that stage did not recognize.

In *Tools for Students Doing Mobile Fieldwork* (Rost & Holmquist, 2008) a series of tools are presented which support data collection and review during field work. The article authors state that students collecting observations while being in the field feel restricted by bringing along equipment such as laptops. Students perceive the equipment to be bulky and that it can be intimidating while interacting with people, such as when conducting an interview. Based on these observations, Rost and Holmqvist have developed a series of tools that aim to provide high mobility during fieldwork and the possibility to better organize observations. During fieldwork, students use an

application running on Symbian-enabled mobile phones to record audio, video, images and text. At some point in time, the recorded data objects are uploaded to a remote repository together with information about who (originating phone) and where (cell tower ID) the data was captured. After the capture process, students use a collaborative workspace (wiki) to browse the recorded data objects. This extension allows content to be browsed either by time of capture or by location. Once the student has found a suitable data object it can be embedded onto the current workspace and further annotated with the tools that the wiki provides.

In the *HyCon* framework (Hansen and Bouvin, 2009), the underlying data model is based on a conceptual context model where it is claimed that context is “defined in general terms rather than specific entities”. The point they make here is that context data is difficult to map directly to a data model due to the generic definition of context (Chen and Kotz, 2000, Dey, 2001). To solve this issue, the data model has been constructed to include the notion of *tagging*. This method allows the base resources to be extended with dynamic properties derived from context, as it is discovered, allowing a very loose data model. Examples of this include tagging resources with GPS coordinates or *2D barcode URLs*, data matrix images that decode to Web resources. The *HyCon* service framework is constructed in four layers: *storage*, *server*, *terminal* and *sensor*. A decomposition of these layers will serve to give a general overview of the interrelations. This framework is illustrated in Figure 2.2 described below:

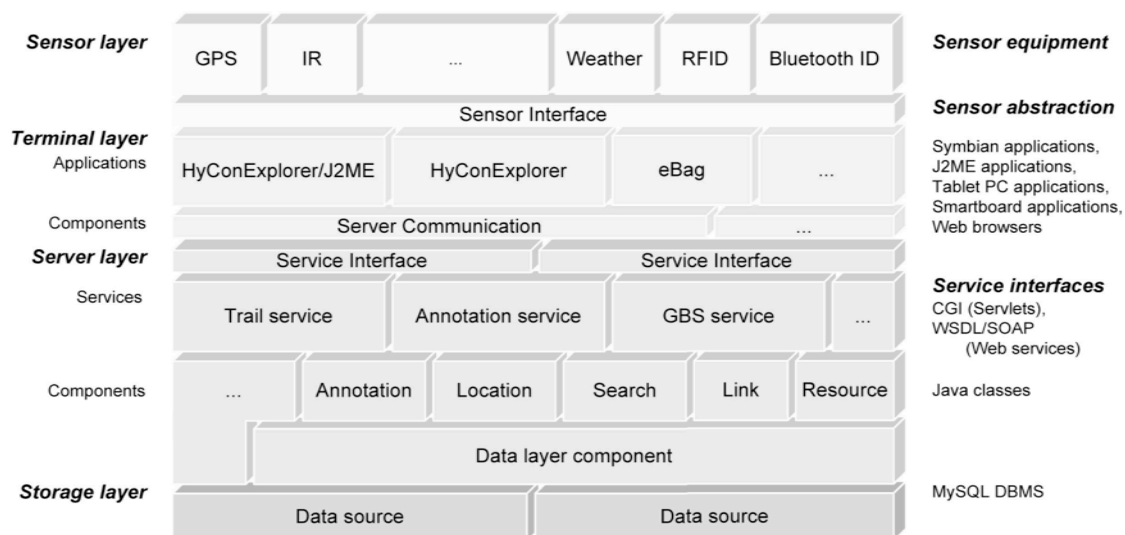


Figure 2.2. HyCon service framework architecture (Hansen and Bouvin, 2009)

The bottom *storage layer* handles persistent storage and retrieval of data. On top of this lies a three-tiered *server layer* where the first tier consists of components, which are composed to services in the second tier. The architecture allows for new components to be added in the framework if there is a need for them, thus also allowing the composition of new services. The third tier of the *server layer*, service interfaces, acts as a communications bridge to the *terminal layer* where applications can interact with the *server layer*. The final *sensor layer* contains sensor abstractions and is presented as decoupled components that hold contextual state in the physical domain. As the state changes these sensor components aggregate data using either a polling mechanism, where applications can request state information, or an event publishing mechanism where observers (applications) are sent notifications about the updated state.

HyConExplorer (Hansen and Bouvin, 2009) was the first prototype built using the *HyCon* framework with the purpose of supporting the activities of collecting, producing

and presenting information from diverse sources. Two versions were developed as *terminal layer* applications: one tailored for tablet PCs and the other for Smartphones. The metaphors employed to support the activities are derived from classic hypermedia interaction seen on the Web with the addition of context awareness. Navigating context-aware hypermedia is accomplished by changing the physical context, which in *HyConExplorer* is performed by moving in physical space or overriding the location sensor and directly interacting with the browsing application. During this, the *HyCon* system looks for contextual cues (such as point in polygon) that it uses to map tagged resources, aggregating them through the *terminal layer*.

Context-aware searching is performed in *HyConExplorer* by inserting search criteria taken from the physical context where again location seems to be predominant. *HyCon* employs a database of all public postal addresses in Denmark, mapped to GPS coordinates to perform an operation known as *reverse geo-coding*: transforming a GPS coordinate to a textual representation of a postal address. In the prototype, all street addresses within a fifty-meter radius of the location sensor are extracted and inserted as search criteria together with keywords specified by the end user. These criteria are then used to execute a search operation via a publicly exposed Web service (Google, 2009a). The results are then visualised onto a map component, which is part of *HyConExplorer*. While users of *HyConExplorer* are moving in the physical space, they are also able to annotate their surroundings. By combining built-in features of mobile devices (camera, microphone) and cheap sensors (GPS receivers) content can be captured and annotated with context properties, previously mentioned as *tagging* the resources. When discussing the implementation of *HyConExplorer* an issue with the *HyCon server layer* is brought forth. The version developed for deployment on Smartphones was tested on Nokia Series 60 Symbian phones and SonyEricsson K750i phones. These do not include support for the Web service interface that the *server layer* exposes. Implications of this were that an additional interface had to be developed to specifically support these phones. This appears to contradict the description of the *HyCon* framework architecture:

... a basic requirement for the design was to allow a large number of heterogeneous clients ranging from simple mobile phones to desktop based applications, Smartboards, interactive floors and Web browsers to be used with the framework.
(Hansen and Bouvin, 2009)

2.2. Literature study summary

Gathered from the results, time and location are important factors to consider when tagging or describing observations. The values by themselves allow for coarse visualization and selection in timeline graphs or in mapping applications, as provided by services such as Google Maps. Aggregate sensor or social network data can be used to further provide contextual information about an object, such as temperature or humidity, after the actual observation has been made. Automated categorization can also provide easy recollection of when and where an observation was made which Naaman et al. also reaffirms.

If the observation includes a digital image, it can be machine analyzed to further improve the accuracy of the automated categorization so that for example a bright image taken at night can be categorized as being “indoors”. This analysis could also include post-processing to discover visual markers that carry associations. Furthermore it can be seen as beneficial to annotate a categorization to indicate if it is human or computer generated, allowing some sort of measurement on how accurate the labelling has been.

The *HyCon* framework can serve as a candidate for comparison against the framework this thesis aims to present. The *HyCon* framework has some issues regarding the server interface, which was identified as part of the literature study. It raises some concerns about how to best integrate the software system with client applications, especially mobile devices that have a restricted environment in comparison to personal computers.

3. Feature decomposition and abstraction

In this chapter, different features from the projects described in the previous chapter will be decomposed into a number of abstract *feature terms* and these are presented first. Thereafter, a number of key architectural patterns, guided by the outcomes from the literature review, will be detailed. The section will close with a summary connecting the projects from the literature study with the architectural patterns and abstracted features.

3.1. Abstracted features

The *location awareness* feature refers to the ability to define exact geospatial properties. It can also include the possibility to perform geometric operations such as measuring the distance between two points or the area of a polygon. In mobile learning software systems the feature is most commonly used as a provider of contextual metadata such as annotating digital media with information that describes where it was captured or created. This feature is also tied to *Team/user awareness* that facilitates the ability to track the location of participants during an activity.

With *team/user awareness* the software system is able to differentiate which actor is performing a certain activity. This can be tied to *storage of user-generated media* where the media is associated with the actor that initiated the storing. *Team/user awareness* can also be used to cordon actors if there is a concern about them interfering with each other in regard to the resources they have created.

Storage of user-generated media allows mobile devices to offload content that has been captured to a content repository. It does not imply that content should transfer to the content repository at point of capture; it could very well be that this is done post-activity by synchronizing over USB or Bluetooth.

Involvement of restricted device(s) is tied to the concern of including devices that are not comparable with standard computers. Restrictions or limitations that can be commonly found are available memory or processing power, restricted communication interfaces, and/or platform concerns regarding application programming interfaces.

Content browsing/reflection involves the feature of providing human actors with user interfaces that allow them to view and make use of the content they have previously stored, to view or make use of human or computer generated annotations or the ability to control the user interface in order to provide the human actor with content during or after an activity. This feature also includes the ability to add content or annotations via a user interface.

The *HTTP service interface* feature is the motivated choice of using HTTP (Hypertext transfer protocol) as a means of providing interoperability regarding communications and data exchange between components.

Content annotation with tags allows resources to be annotated with properties that are created by human actors or the system it self. The action to annotate with a tag is either initiated as part of the feature provided by *storage of user-generated media*, *content browsing/reflection* or the feature of *post-processing*. The flexibility of this feature is highly dependant on the data model design and therefore can be associated with the feature of *heterogeneous data*.

Post-processing is the ability of performing work on a resource where the action is not directly initiated by the user. This can include image processing to discover visual markers (creating resource associations), image processing for the purpose adding contextual attributes that are not immediately apparent (such as facial recognition) or

processing digital content for the purpose of extracting embedded metadata and annotating resources with such properties. It can relate to the feature of *event awareness* in the fact that the post-processing components can be passively observing the software system until an event initiates the work. This is especially apparent in an environment where the user is interacting with the system using a web browser. The nature of the hypertext transfer protocol (HTTP) protocol dictates that there is a maximum time allotted for the request/response cycle. If the HTTP server component must perform time consuming or processing intensive work this could lead to severe performance issues such as thread pool exhaustion (where there are too many open connections). To alleviate this, the web server component can emit an event message that specifies the job that must be performed. *Post-processing* components listen for these event messages, perform the work and then emit a new message informing that the work is complete.

The feature of *event awareness* allows components in the software system to broadcast events while not being fully dependent on if the message has been delivered or not. The feature can facilitate the decoupling of components in the software system, allowing them to be added or removed as the system is running.

In an environment where sensors are being utilized the *event awareness* feature is highly beneficial. Without it client components must continuously poll sensor interfaces, querying for updated sensor data. If a communications channel cannot be established or is broken it is difficult to determine if the sensor has failed or if it is just temporarily unavailable.

Heterogeneous data is similar to *content annotation with tags* dependant on the data model design. Allowing the feature is a decision that must be taken at the early stage of the project lifecycle. By introducing this feature in the data model a higher degree of flexibility is ascertained regarding the object typing. A side effect of the feature is the requirement of having extra programming logic in place to determine the typing of an object. Loosely typed data objects are more flexible but require type hinting to describe what kind of object is being referenced, a requirement that strongly typed data objects do not have to the same extent.

3.2. Connecting with architectural patterns

Gathered from the literature study results two key architectural patterns can be identified and will be presented in the following two subsections.

3.2.1. N-tiered architecture

The n-tiered architecture, illustrated in Figure 3.1, is best explained as progressively breaking down an application into several interconnected chunks, or tiers. The first tier is indicative of a non-networked, self-contained application that the user interacts with. Adding the second tier lets the application be (network) distributed, transferring all concerns except presentation from the first tier.

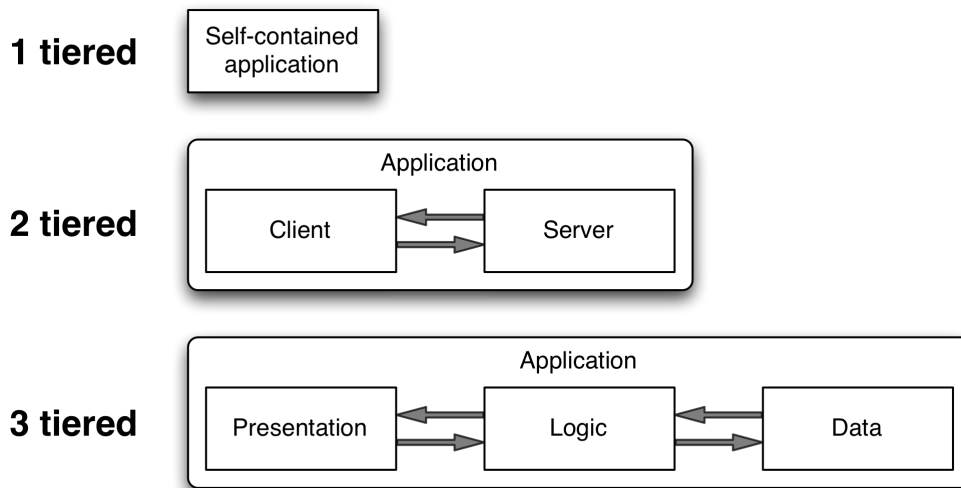


Figure 3.1. A N-tiered architectural pattern

This architectural style naturally only starts applying when separating an application into two tiers. An example of this style is where a client application executes queries against a database server and presents the results of the query to the user. The distributed nature allows multiple clients to be connected to the database simultaneously, which is not possible in a one-tiered architecture.

The three-tiered architecture separates data from logic in the second tier. The logic tier has many different concerns such as caching data or providing secured access, also known as *middleware* (Fowler, 2000).

3.2.2. Event-driven architecture

In the event-driven architecture, illustrated in Figure 3.2, events are emitted as they are initiated by sources, which can be any number of things that are aware of the event generator. Typically we can find sources such as data stores, sensors or business processes (as initiated by the user or the application). Events flow thru the event channel, visiting any event processor that is connected to the event channel. Key to this architectural style is that it is not required that there should be event processors connected to the event channel at all times, allowing the system to be highly decoupled.

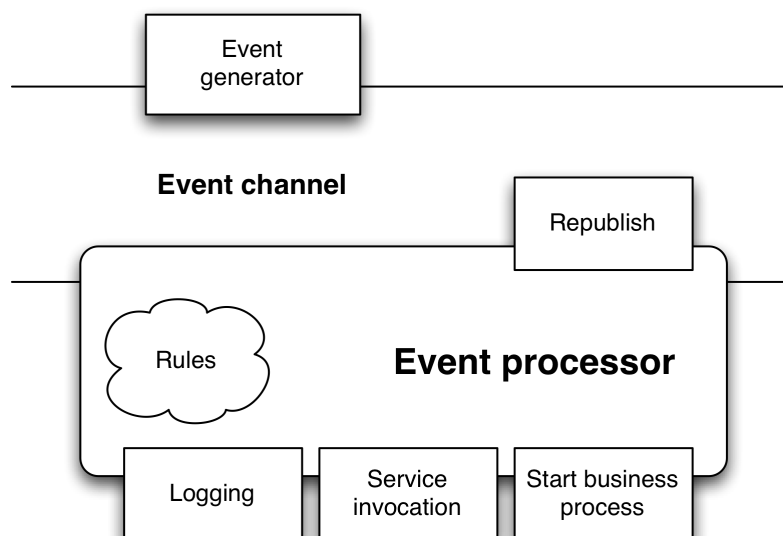


Figure 3.2. An event-driven architecture

The event processor is allowed to decide if it should act on an event or not. If so, rules that are provided to the event processor determine what action(s) an event could initiate. Figure 3.2 shows common actions when events are processed. For example, a sensor could emit an event that also carries the sensor state as payload. The event processor being interested in such events logs the sensor state to a data store, determines if it should invoke a remote service such as an alarm (a sensor threshold has been exceeded) and/or initiates a business process that transforms the event payload, possibly republishing a correlated or an entirely new event to the event channel.

3.3. Summary

The previous section presented abstracted features from the projects included in the literature study from section 2. Figure 3.3 presents a matrix associating these features to projects from the study. From this matrix we can draw an initial conclusion indicating a reaffirmation that there indeed are many shared aspects concerning the software systems from the study. A noteworthy comment is that P4 could potentially also be associated with the feature of *location awareness* as they mention it in their upcoming work.

	P1	P2	P3	P4	P5	P6
Location awareness	✓	✓	✓		✓	✓
Storage of user generated media		✓	✓	✓	✓	✓
Team/user awareness	✓		✓	✓		✓
Involvement of restricted device(s)			✓	✓	✓	✓
Content browsing, reflection		✓		✓	✓	✓
HTTP service interface		✓	✓		✓	✓
Content annotation with tags		✓	✓			✓
Post-processing		✓	✓	✓		
Event awareness	✓					✓
Heterogeneous data		✓		✓		

P1 Ambient Wood
 P2 PhotoCompas
 P3 ZoneTag
 P4 ButterflyNet
 P5 Tools for Students Doing Mobile Fieldwork
 P6 HyConExplorer

Figure 3.3. Feature matrix

Connecting back to the architectural patterns presented in 3.2, we can observe a predominant applicability of the n-tiered architecture and to a less extent the event-driven architecture. In a similar fashion as the previous feature matrix, an architectural pattern matrix is presented in Figure 3.4.

	P1	P2	P3	P4	P5	P6
N-tiered	✓	✓	✓	✓	✓	✓
Event-driven	✓			✓		✓

P1 Ambient Wood
 P2 PhotoCompas
 P3 ZoneTag
 P4 ButterflyNet
 P5 Tools for Students Doing Mobile Fieldwork
 P6 HyConExplorer

Figure 3.4. Architecture matrix

It should be noted that P3 and P5 have been excluded as event-driven architectures but do to some extent display event-driven *behaviour* in the fact that they both employ a background process that monitors changes in network topology (switching to another cell tower). They both miss the notion of an event channel and indirectly also the possibility for other event processors to act upon events, which are necessary parts of an event driven architecture.

4. Validation

This section will begin by presenting the requirements that were assimilated from the study and feature extraction of the previous section. The outcome of the requirements engineering process will be merged with the two key architectural patterns of the previous section to construct a number of conceptual architectures to act as guidelines when developing the framework implementation. After this we will find details on the implementation done as part of this thesis together with aspects related to testing the implementation. At the closure of this section we will find the implemented framework being applied in two case studies to explore adaptability and extensibility aspects.

4.1. Requirements engineering

Requirements engineering is an important process where the capabilities and boundaries of the final system is defined (Arlow & Neustadt, 2005). They are essentially divided into two types:

1. **Functional requirements.**
Defines what behaviour the system shall offer.
2. **Non-functional requirements.**
Defines constraints on the system.

Requirements should only say what a system should do but not detail how it should do it. That being said the requirements that are presented also include details on how the final implementation should fulfil some requirements. This is not uncommon since it makes it easier to understand the implementation (Arlow & Neustadt, 2005).

Based on the features discovered as part of the literature study and the feature extraction process, a list of requirements (included in appendix A) is compiled and prioritised according to the MoSCoW (Dai Clegg and Barker, 1994) technique (abbreviations presented in Figure 4.1). The prioritisation timeframes the requirements, defining what must be included in the deliverable and what can wait for future revisions. For the sake of brevity, short hand notation will not be used. The lists of requirements have the identifiers *FR* (functional requirements) and *NFR* (non-functional requirements).

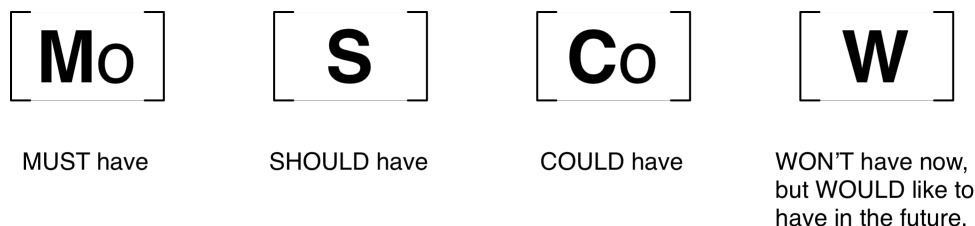


Figure 4.1. MoSCoW prioritisation

Referring to the appendix A, the requirements are engineered for a system that under most circumstances includes additional requirements, which add project specific aspects. As such FR:1, FR:2 can only be considered against the system without project specific extensions.

FR:3 is an optional requirement to facilitate data visualisation or manipulation from external applications. We can find a coupling to NFR:3 here in the fact there exists a broad spectra of web applications that can consume, analyze and visualize machine readable documents (Yee, 2008).

FR:4 is essential as was discovered in section 3 where the feature of *location awareness* existed in almost all projects and can be connected with FR:6. NFR:4 enables this requirement to great extent.

NFR:7 is a desirable but not necessary requirement which allows the final system to build upon existing abstractions, saving time spent during development.

NFR:3 concerns the involvement of restricted devices and the limitations previously mentioned in section 3. By constructing the service interfaces with device restrictions in mind it would also be applicable to non-restricted devices. By ignoring the requirement several interfaces have to be constructed conforming to varying levels of restrictions or capabilities. As the amount of interfaces grows, so does the cost for developing and testing them.

NFR:5 is desirable since it presents up-to-date documentation for developers. In addition to testing software, unit tests can serve to demonstrate system functionality and should be created as part of the development process.

4.1.1. Conceptual system architectures

The most basic architecture that can be composed from the requirements is the *content repository*. It is a fundamental system architecture that exists in many Mobile Learning software systems which we saw in section 3.2. The architecture builds from the n-tiered architectural pattern and is most commonly implemented as a HTTP client/server application and a relational database management system in the third tier. The middleware is either an application server (such as J2EE application servers or Internet Information Services from Microsoft) or a web server with support for high level scripting languages (such as Apache HTTP server). In an implementation such as the *content repository*, it must provide a client authentication (but not necessarily authorization) mechanism associating who created what content or cordon access to resources. The data adapter is an abstraction that can allow more programming language natural interaction with the database.

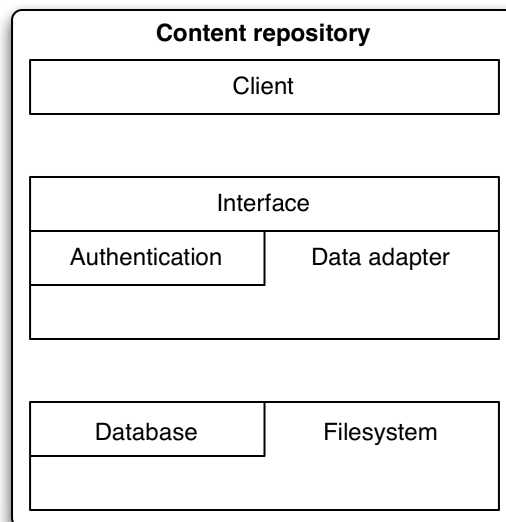


Figure 4.2. Content repository

In an environment where sensor equipment cannot directly communicate with applications the *sensor net proxy* (Figure 4.3) is a viable system architecture. The main purpose of the architecture is to bridge two incompatible transport interfaces via the message bus. The architecture additionally can allow persistent messages, which is useful if it cannot be guaranteed that message recipients are always connected to the

message bus. As subscribers (re-) connect to the message bus, persisted messages are delivered to their destination recipients.

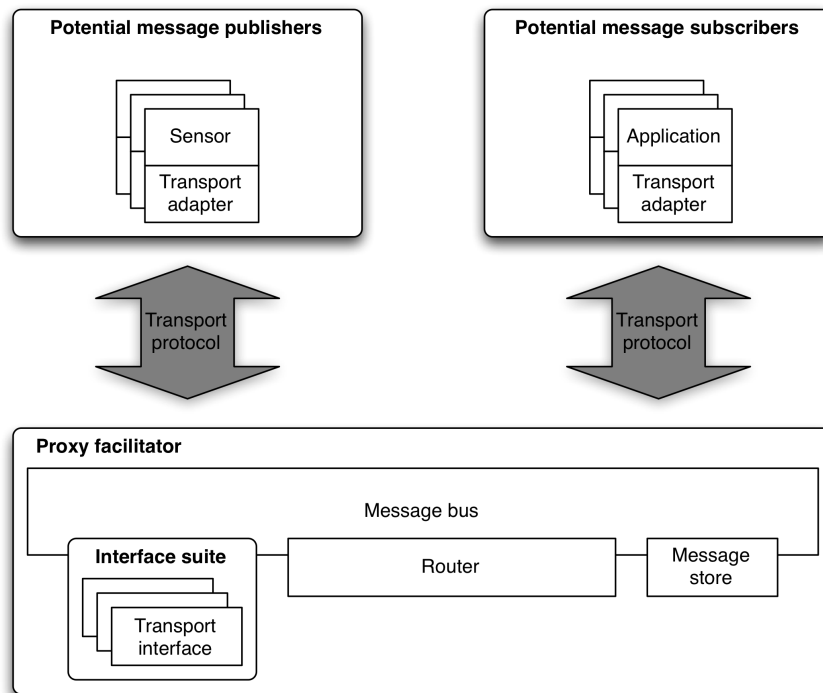


Figure 4.3. Sensor net proxy

The message publishers in the *sensor net proxy* can be any kind of device capable of exposing sensor information over a communications channel. The device must either natively be able to adapt to a transport protocol or include some feature that enables a transport protocol adapter to be implemented. The routing component in its simplest form has no rules and allows all messages to pass thru the bus. A more complex router will inspect the contents of a message and determine which subscriber is the correct recipient, indirectly addressing security and performance concerns within the architecture.

Composing the *content repository* and *sensor net proxy* architectures can generate architectures that are interesting to the problem domain of this thesis. Presented in Figure 4.4 is the *location aware activity system*. This architecture correlates real-time data provided by the location sensors with actors within an activity. As actors are moving in the physical space the *content repository* (extended with an activity management system) is updated to provide the presentation tools with content related to the space that the actors occupy. It can be implied from the figure that the *proxy facilitator*, activity management system and *content repository* acts as a single architecture but this is not necessarily the case. The sensor/actor correlation mechanism can exist in either of the two components, or be a completely separate system, but naturally belongs to the activity management system component due to the dependency of the data tier (which holds information about actors in the activity).

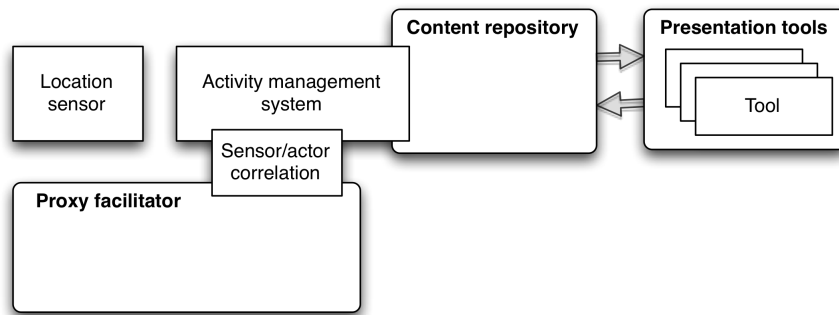


Figure 4.4. Location aware activity system

Another interesting composition of the *content repository* and *sensor net proxy* (specifically the proxy facilitator) is the *event aware content aggregation* (Figure 4.5). By using a database implementation capable of acting as an event generator, content can be aggregated as it is created or updated by the content creation tools. The destination of the aggregated content can be any kind of web application that exposes an application programming interface or is otherwise aware of the proxy facilitator and thus able to poll for new information. One such instance, where an application programming interface is exposed, are the services provided by Google (Google, 2009b). Taking for example, an environment such as *ButterflyNet* (Yeh et al., 2006), the content creation tools could be used to store tabular data in the content repository. As content flows into the repository, it is aggregated to a Google Spreadsheets (Google, 2009c) document and presented in near real-time.

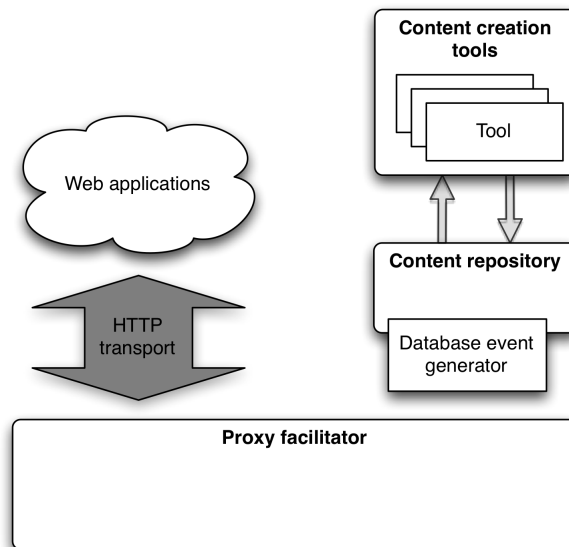


Figure 4.5. Event aware content aggregation

4.2. Implementation and testing

The framework implementation (regarding choice of software) will to some extent be influenced by previous work that has been reported in (Milrad, 2008), referred to as the *Karamel application*. The software components used for this implementation have been intentionally selected due to their successful use in prototypes or small-scale projects in my related work. The intentional choices will also safeguard that future work can be conducted without major changes to the code base.

I will first present the rationale of the framework data model as adopted in the implementation. Following this, I present some relevant terminology before detailing

the framework implementation. At the end of this chapter, I discuss and present results from the test scenarios that have been used during the development process.

4.2.1. Framework data model and rationale

Some terminology presented in the data model (see appendix B) includes implementation specific details that will be expanded upon later in section 4.2.2. A simpler analysis diagram (see Figure 4.6) is used in order to illustrate the rationale behind the proposed data model.

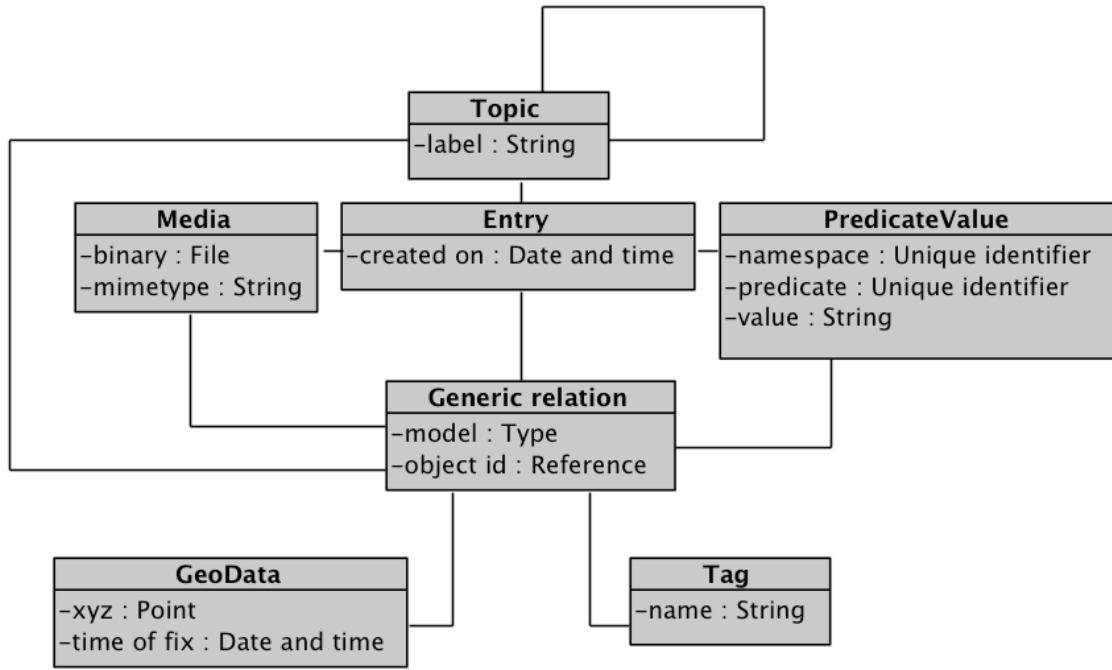


Figure 4.6. Framework analysis diagram

A *Topic* is a container of *Entries* and can associate to itself to create hierarchies of *topics*. *Entries* are outer containers for *media* objects and metadata via *PredicateValue* objects. A *PredicateValue* is a triplet of the subject (association with *Entry*), the predicate (predicate property) and the object (value property). The namespace property is used to distinguish two similar predicates. For example, if using temperature as the predicate and a number as value it is indeterminate what unit the value signifies. By providing the namespace “Celsius” the unit can be determined.

The *Media* model is a generic container of media-like objects (ranging from images to plain text documents). The *Generic relation* model is concerned with associating *GeoData* and *Tag* objects to any of the models previously mentioned. That is, if a *Media* carries an association to *Generic relation* it can also associate to a *GeoData* object via the model and object id properties. Putting this in context with a relational database, the type property references a table and the object id property references a primary key in that table. The point property of *GeoData* objects is a triple of latitude (x), longitude (y) and altitude (z).

Tag objects are used to label any of the previous models and also it self. A *Tag* is unique and can be used as a multiple association. That is, one *Tag* can be associated with any number of model instances and a tagged instance has a link to any other tagged instance as long as they share the same *Tag* object.

4.2.2.Relevant terminology

Before proceeding with the details of the implementation some relevant terminology will first be presented.

Web application frameworks

The web application framework serves as a foundation where problems and tasks related to the development of web applications have been abstracted. Referring to the n-tiered architecture (three tiered) previously presented in section 3.2.1, the web application framework exists in the middle layer. A list of terminology will serve to define abstractions commonly found in this kind of framework.

Object-relational mapper.

The purpose of the object-relational mapper (ORM) is to enable interaction with a relational database from an object oriented language. The mapping can be generated either by database schema introspection, that generates code stubs, or by modelling the database using program code which when executed generates an database importable schema definition.

Caching.

Web application frameworks can include methods to control how and if a resource should be cached or not. This improves overall performance by eliminating requests for resources that seldom change, such as images to some extent.

Scaffolding.

A substantial amount of development time is spent constructing user interfaces for manipulating the database. The ORM can as part of the mapping process generate user interface scaffolds for these purposes.

Routing.

A front-controller allows an HTTP request to be mapped to a method defined by the application developer. It is not uncommon to find front-controllers that inspect the content of the request, mapping relevant information to parameters in the method call.

Templating.

To separate data representation (via user interfaces or other) from application logic, a templating engine is often part of a web application framework. The engine is fed with an initial template document and a data context. Using template specific syntax, the context is accessed and merged to the final representation. This separates programmer and designer concerns in the fact that user interface designers create the template documents and in agreement with the programmer is aware of what the data context provides.

Authentication, authorization and session management.

Similar to scaffolding, a substantial amount of development time is spent implementing secured access to a web application and keeping the client state (session tracking), Therefore, this process is often abstracted with a web application framework. It is also common to see a component-based approach to the abstraction allowing different authentication/authorization back-ends to be used.

Publish/subscribe event system

This kind of system is fundamental to the event-driven architectural pattern (described in section 3.2.2) and I find it as the message bus of the *proxy facilitator* as presented in section 4.1.1. A predicate is used to determine how a message should be routed between event publishers and event subscribers (Huang & Garcia-Molina, 2004). A number of subscription models exist (Baldoni et al., 2009) and we will only focus on the *topic-*

based model as it is most pertinent to the framework implementation this thesis presents.

In the *topic-based model* subscribers declare an interest for a particular topic (the predicate). As the publishers generate events, they attach a *topic label* to the event. When the router receives the event, it inspects the labelling and routes the message accordingly. In the model we can both find flat and hierarchical structuring where the hierarchical structure allows subscription to sub-topics. That is: if *TopicB* is a sub-topic of *TopicA*, subscribers to *TopicA* will receive events sent to *TopicA* and *TopicB*. The model can also cater for wildcard topics: if there is a subscription to *TopicA* * all events sent to *TopicA* and below will be eligible for delivery. Extending the model with such hierarchical structuring allows for finer granularity compared to the flat structure (Baldoni et al., 2009).

4.2.3. Implementation details

The implementation has been programmed using *Python* (Python Software Foundation, 2009) and the *Django* (Django Software Foundation, 2009) web application framework. *PostgreSQL* (PostgreSQL Global Development Group, 2009) has been used as the database management system, extended with *PostGIS* (Refractions Research, 2009) to enable geographic objects and operations. *MorbidQ* (OSSLine, 2008) was used to facilitate an event publish/subscribe system in the implementation. Events are published to an instance of *MorbidQ* via *PostgreSQL* (indirectly from the *Django* application) as database objects are created, updated or deleted. This is facilitated via *PostgreSQL trigger functions* (source code found in appendix F) that are attached to database objects as the database schema is imported. The data matrix image analysis was facilitated by *libdmtx* (Laughton, 2009) and associated *Python* wrapper libraries.

The choice of software stems from the requirements engineering process in section 4.1 and in particular that *Django* integrates very well with *PostgreSQL* and implicitly also *PostGIS* to enable geospatial features. *MorbidQ* was selected due to it being implemented in *Python* and as such allows runtime access to *Django* subcomponents (in particular the object-relational mapper). *Libdmtx* was selected purely on convenience as there are *Python* wrappers available and thus, similar to *MorbidQ*, allows easy integration with *Django* and its subcomponents. Relating to the system requirements detailed earlier, none of the software selected for the implementation is strictly required but *PostgreSQL* and *PostGIS* should be considered as highly desired components.

At the current time, the implementation includes the framework data model expressed as *Django database models* (source code found in appendix C) and database event generators. The application service interfaces build on URL routing capabilities from *Django* and attempts to adopt the resource oriented architectural guidelines presented by (Richardson & Ruby, 2007). Adopting these guidelines exposes a *uniform* service interface that does not discriminate what type of client is accessing the service which is not the case with other types of service interfaces such as SOAP over HTTP (Prescod, 2002). To some extent this addresses the concern discovered in the literature study regarding the *HyCon* service interfaces.

4.2.4. Unit test results

The implemented framework currently has 9 test cases, which decompose to 14 unit tests. Six test cases are put in place to do functional testing of the data model and the final three test cases are used to test specific features of the framework implementation:

1. ProxyModelTest

Tests and demonstrates how heterogeneous data (via the *PredicateValue* model)

can be strongly typed at run-time, in this case to a *TemperatureMetadata* model which additionally is extended with business logic to do unit conversion.

2. **GISTest**

The test imports a dataset containing polygon definitions of world borders and is used for intersection testing. The test uses three arbitrary locations, as a set of *GeoData* objects, from Finland, Norway and Sweden. Each *GeoData* object is used to query the world border dataset, where the result of each query is a polygon labelled with the corresponding country name. The successful test is if all countries are found.

3. **DataMatrixTest**

Tests that the database correctly generates events, which is performed by letting a worker application (source code found in appendix G) attach to the message bus for the purpose of post-processing an image to detect visual markers.

Additionally the worker application associates the post-processing results with the *Media* object (event source) via a *PredicateValue* object. The image used for analysis can be found in appendix H.

Appendix D shows a verbatim copy of the console after executing the unit tests, demonstrating no inconsistencies. The long runtime of the tests is due to the world borders dataset being loaded into the database. Source code for the unit tests can be found in appendix E.

4.3. Case studies

In the following two subsections, the proposed thesis framework will be applied to two case studies in order to demonstrate its adaptability and extensibility. The discussions will to some extent be based on the framework implementation described in the previous section.

4.3.1. Framework adaptability aspect

During the LETS GO (Learning Ecology with Technologies from Science for Global Outcomes) project trials in 2009 a software system (concept in Figure 4.7) was built to support the activities. This project is an on-going collaborative effort between Stanford University, CeLeKT, PASCO and schools in Sweden and USA (more information about this project can be found at: (CeLeKT, 2009)). One of the trials' scenario involved groups of students in the field observing trees, collecting data and annotating their findings using various sensors and digital media capture devices. Back in the classroom, they reflected and discussed upon the activities conducted in the field by visualizing the data they collected.

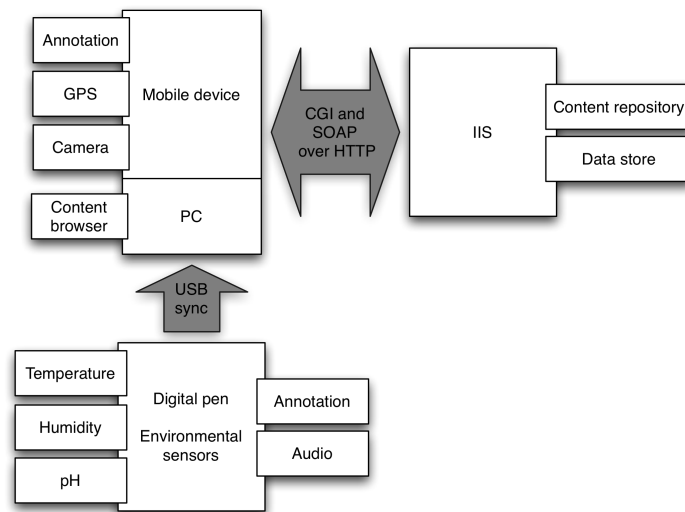


Figure 4.7. LETS GO system concept

In this case, the framework can be applied as a *content repository*. The framework data model, which was discussed in 4.2.1, can be said to be sufficiently adaptable (*PredicateValue* model demonstrated in 4.2.4) to capture all the environmental values important to the activity. The framework data model also allows content association to groups via the *Topic* model and indirectly via the *Entry* and *Media* model.

In the LETS GO content browser, time was used to correlate photographs taken during the activity (see *Timeline display* indicator of Figure 4.8), which again can be tied to the *Entry* and *Media* model. An interesting aspect would be the inclusion of visual markers in the photographs. This could allow the framework *Tag* model to be used as a grouping mechanism. From the diagram described in Figure 4.8, we can also find content filters (left side) that the *Tag* model naturally captures. In the bottom we find a map control associating the sensor data with geographic locations, connecting with the framework *Entry* and *GeoData* models.

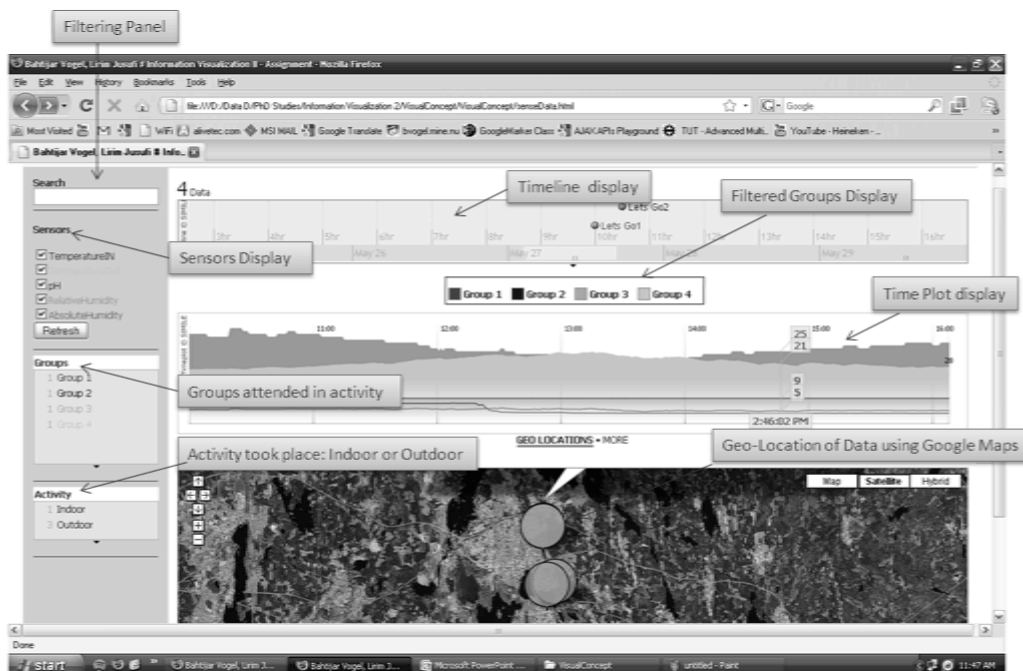


Figure 4.8. LETS GO content browser

4.3.2. Framework extensibility aspect

The Geometry Mobile (GeM) project aims at promoting collaborative learning in the field of mathematics using augmented reality tools, facilitated via mobile devices, web applications and 3D visualisation tools. The learning objectives allow middle school students to explore geometric concepts such as height, perimeter, area and volume that they apply to construct a virtual building on the Våxjö University campus. More detailed information about these activities can be found at (Ung Kommunikation, 2009). The initial trials conducted in spring 2009, presented one sub-activity where the students were directed to an area where they were instructed to first guess the diameter and then measure it accurately using three mobile devices: two as location providers (via GPS) and the third as a presentation tool, illustrated in Figure 4.9.

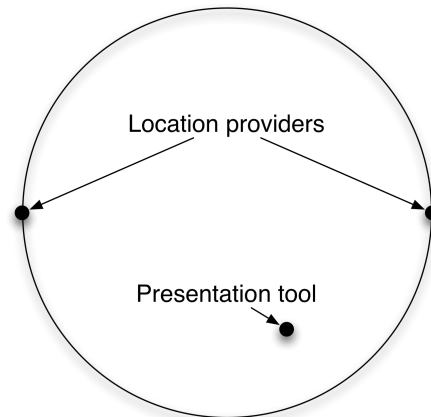


Figure 4.9. GeM sub-activity concept

In this instance, the mobile devices had an embedded web server with scripting extensions installed. A background PyS60 (Python for S60 platform) process acted as a bridge between the built-in GPS sensor and the embedded web server as it is not possible to access sensor data from the web server process (more specifically the scripting extensions). The bridging was facilitated by a database that was shared between the web server application and the background PyS60 process. This database was updated every few seconds with current location data. To measure the distance, the presentation tool pulls location data from each location provider over HTTP and performs calculation over the provided latitude/longitude pairs. In retrospect, it can be seen that there are many components put in place to address issues that essentially are found with the embedded web server. In this particular case, the thesis framework implementation can to some extent act as a *sensor net proxy* (presented in section 4.1.1). Figure 4.10 shows this architecture adapted for the scenario just described.

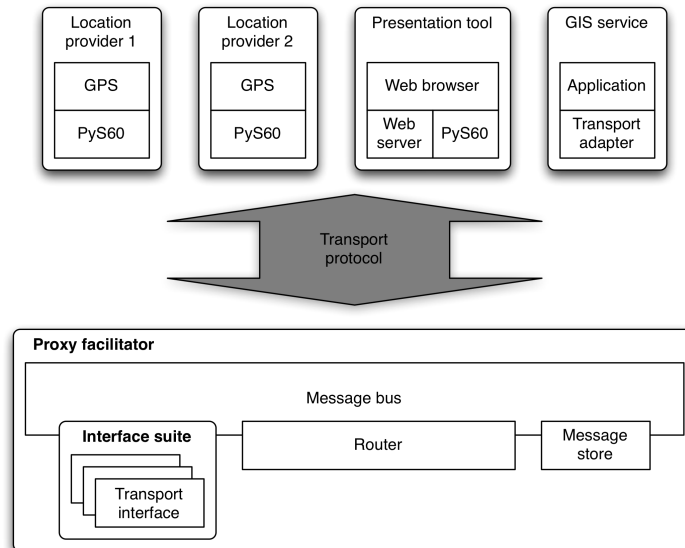


Figure 4.10. GeM sensor net proxy

The two location providers are modified to connect with the message bus at the socket level (adapts via a PyS60 application), allowing a persistent communications channel to the other components in the architecture. The presentation tool is modified to query itself (localhost) instead of polling the remote location providers and also includes a transport adapter similar to the one that the location providers have. Finally, we introduce the new *GIS service* (framework) component that is concerned with performing the distance measurements. Figure 4.11 illustrates the event schema that will be referred to when detailing the event flow.

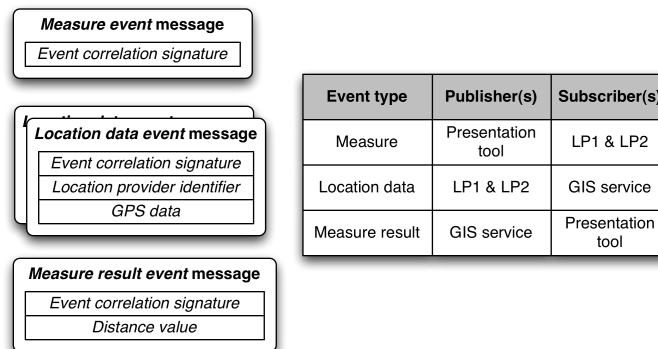


Figure 4.11. Event schema

The presentation tool acts as the *measure event* source and is initiated by the user browsing to a web server script application resource. Before the event is propagated to the message bus, the presentation tool script generates an *event correlation signature* and attaches it to the event message. As the message enters the bus, the other three components act asynchronously: the two location providers each emit a *location data event* message attaching sensor identifiers (LP1 and LP2 respectively) and GPS data. The *GIS service* receives all events but does not act until it has seen all three (using the *event correlation signature* to form associations). Upon having received all three events, the *GIS service* performs work and emits a *measure result event* message that the presentation tool has subscribed to, ending the flow of events. PyS60 has access to underlying operating system features and can thus present a notification window with the results of the measurement to the user.

5. Conclusion

This section presents the results and conclusions that can be drawn from the work that has been presented in this thesis. The section will close with elaborations on potential future directions of my coming work.

5.1. Results

This thesis set out to explore the field of mobile learning and the architecture patterns and software components within it, for the purpose of optimizing the software engineering process in new or repurposed software systems. To accomplish this goal, I looked first at a number of existing software systems, studying their individual features and abstracting them to manageable definitions and associating these software systems with two key architectural patterns.

Based on the findings, a set of requirements was formulated and I applied them in several conceptual software architectures. To validate the findings, the requirements have been applied to an implementation built using a sophisticated web application framework and many technologies that can be related to the conceptual architectures. I tested the implementation using a software-testing framework where the test cases to some extent could be related to the different software systems we initially looked at. I also applied the implementation to two case studies, discussing aspects related to adaptability and extensibility.

The thesis problem was stated as *how can we optimize the software development process by pre-emptively identifying key features common in software systems as used to support learning activities and field observation with mobile devices?* Taking this in retrospect, it can be said that the results of the work in the thesis address many of the commonalities found in mobile learning software systems but referring to the limitations set initially it is quite possible that there are more commonalities to be found. The results in the first part of the thesis can act as guidelines, when initiating a new project having in mind that the results presented here are based on a broad spectrum of software systems. The second part of the thesis follows to some extent these guidelines and presents an implementation that is adaptable and extensible as shown in the case studies.

There is a concern with data representation and/or data visualisation, but it can be said to be very tied to project specific needs and thus hard to capture in a general framework such as the one presented here. Relying on the requirement of using a web application framework, somewhat addresses the concern by the provision of tools that simplify the development of web based user interfaces. Another concern can be raised about using a relational database to act as the data store. The relational model relies on having a well-defined schema and does not adopt well for heterogeneous data. The data model presented in the work circumvented this with the *PredicateValue* model but has a side effect of not being able to index such data effectively as we no longer express in (database) schema what kind of data is being referred to.

5.2. Future work

An interesting venue for future work would be to include more projects in the literature survey in order to find more services that can attach to the message bus, such as the *GIS service* component from section 4.3.2. Another possible direction could be to replace the *rigid relational* database with a more expressive object/document databases. If we look at the data model rationale and how the *PredicateValue* object was defined we see a close relationship to the Resource Description Framework (RDF) and how it defines

an RDF statement as a triple of subject, object and predicate. Some work in this direction has been done (recently reported in (Svensson et al., 2009) and it would be interesting to explore how the work done in this thesis could be adapted to such an environment.

As mentioned in the results and implementation section, no efforts have been made to implement data representation or visualisation, which in effect currently leaves the framework implementation without any user interfaces and as such can be seen as an entry point for my future work.

References

- Ahern, S., Davis, M., Eckles, D., King, S., Naaman, M., Nair, R., Spasojevic, M. & Yang, J. (2006). Zonetag: Designing context-aware mobile media capture to increase participation. *Proceedings of the Pervasive Image Capture and Sharing Workshop (PICS'06)*,
- Arlow, J. & Neustadt, I. (2005). *UML 2.0 and the Unified Process: Practical Object-Oriented Analysis and Design*, Addison-Wesley.
- Baldoni, R., Querzoni, L., Tarkoma, S. & Virgillito, A. (2009). Distributed Event Routing in Publish/Subscribe Systems. *Middleware for Network Eccentric and Mobile Applications*. Springer Berlin Heidelberg.
- CeLeKT. (2009). Center for Learning and Knowledge Technologies. *LETS GO: Learning Ecology with Technologies from Science for Global Outcomes*. Available: < <http://www.celekt.info/projects/show/20> > (2009-11-13).
- Chen, G. & Kotz, D. (2000). A survey of context-aware mobile computing research. Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College.
- Dai Clegg, R. & Barker, B. (1994). Case Method Fast-Track: A Rad Approach. Addison-Wesley Longman Publishing Co., Inc., Boston, MA.
- Dey, A. (2001). Understanding and using context. *Personal and ubiquitous computing*, Vol. 5, P. 4-7.
- Django Software Foundation. (2009). *Django | The Web framework for perfectionists with deadlines*: Available: < <http://www.djangoproject.com> > (2009-11-13)
- Druin, A. (2009). *Mobile technology for children: Designing for interaction and learning*, Morgan Kaufmann Pub.
- Fowler, M. (2000). *Analysis Patterns: reusable object models*, Addison-Wesley.
- Frohberg, D., Goth, C. & Schwabe, G. (2009) Mobile Learning projects-a critical analysis of the state of the art. *Journal of Computer Assisted Learning*, Vol. 25, P. 307-331.
- Google. (2009a). *Google AJAX Search API - Google Code*: Available: < <http://code.google.com/apis/ajaxsearch> > (2009-09-06)
- Google. (2009b). *Google Data Protocol - Google Code*: Available: < <http://code.google.com/apis/gdata> > (2009-09-19)
- Google. (2009c). *Welcome to Google Docs*: Available: < <http://docs.google.com> > (2009-09-10)
- Hansen, F. A. & Bouvin, N. O. B. (2009). Mobile Learning in Context—Context-aware Hypermedia in the Wild. *International Journal of Interactive Mobile Technologies (iJIM)*, Vol. 3, P. 6-21.
- Huang, Y. & Garcia-Molina, H. (2004) Publish/subscribe in a mobile environment. *Wireless Networks*, Vol. 10, P. 643-652.
- Jacobson, I., Booch, G. & Rumbaugh, J. (1999). *The unified software development process*, Addison-Wesley.
- Laughton, M. (2009). *libdmtx Home*: Available: < <http://www.libdmtx.org> > (2009-11-13)
- Milrad, M. (2008)- *Data Portability and Media Migration for the Mobile Internet (MeMiMo) Final Report*: Available: < http://iis.se/docs/Final_Report_08MeMiMo_reduced.pdf > (2009-09-19)
- Naaman, M., Harada, S., Wang, Q., Garcia-Molina, H. & Paepcke, A. (2004^a). Context data in geo-referenced digital photo collections. *Proceedings of the 12th annual ACM international conference on Multimedia*, New York, USA
- Naaman, M., Song, Y. J. S., Paepcke, A. & Garcia-Molina, H. (2004^b). Automatic organization for digital photographs with geographic coordinates. *Proceedings*

- of the 4th ACM/IEEE-CS joint conference on Digital libraries, Tuscon, AZ, USA
- OSSLIne. (2008). *MorbidQ*: Available < <http://www.morbidq.com> > (2009-11-13)
- PostgreSQL Global Development Group. (2009). *PostgreSQL: The world's most advanced open source database*: Available: < <http://www.postgresql.org> > (2009-11-13)
- Prescod, P. (2002). Roots of the REST/SOAP Debate. *Proceedings of Extreme Markup Languages Conference*, Montréal, Canada
- Python Software Foundation. (2009). *Python Programming Language -- Official Website*: Available: < <http://www.python.org> > (2009-11-13)
- Refractions Research. (2009). *PostGIS : Home*: Available < <http://postgis.refractions.net> > (2009-11-13)
- Richardson, L. & Ruby, S. (2007). *RESTful web services*, O'Reilly.
- Rogers, Y., Price, S., Fitzpatrick, G., Fleck, R., Harris, E., Smith, H., Randell, C., Muller, H., O'malley, C., Stanton, D., Thompson, M. & Weal, M. (2004). Ambient wood: designing new forms of digital augmentation for learning outdoors. *Proceedings of the 2004 conference on Interaction design and children: building a community*, Maryland
- Rost, M. & Holmquist, L. E. (2008). Tools for Students Doing Mobile Fieldwork. *Fifth IEEE International Conference on Wireless, Mobile, and Ubiquitous Technology in Education*, Beijing, China
- Sharples, M., Milrad, M., Arnedillo-Sánchez, I. and Vavoula, G., (2008). "Mobile learning: Small devices, big issues," in Balacheff, N., Ludvigsen, S., de Jong, T., Lazonder, A., Barnes, S., and Montandon, L., (eds.) *Technology Enhanced Learning: Principles and Products*, Springer, Berlin, Germany.
- Spikol, D. & Milrad, M. (2008). Physical Activities and Playful Learning Using Mobile Games. *Research and Practice in Technology Enhanced Learning*, Vol. 3, P. 275-295.
- Svensson, M., Pettersson, O. & Persson, M. (2009). Pinetree: A Learning Content Repository Based on Semantic Web Technologies. *Ninth IEEE International Conference on Advanced Learning Technologies*, Riga, Latvia
- Ung Kommunikation. (2009). *Amulets - en temagrupp inom Ung Kommunikation*: Available: < <http://amulets.blogg.se> > (2009-11-13)
- Yee, R. (2008). *Pro Web 2.0 Mashups: Remixing Data and Web Services*, Apress.
- Yeh, R. B., Liao, C., Klemmer, S., Guimbretière, F., Lee, B., Kakaradov, B., Stamberger, J. & Paepcke, A. (2006). ButterflyNet: a mobile capture and access system for field biology research. *Conference on Human Factors in Computing Systems*, Montréal, Canada

Appendix A : System requirements.

Identifier: FR Name: System functional requirements.	
FR:1	The system shall provide user interfaces for browsing model instances.
Priority	Must have.
Description	Content needs not only be stored but also browsed by a human agent.
FR:2	The system shall provide user interfaces for creating or updating resources.
Priority	Should have.
Description	It cannot be assumed that all content is stored by a computer agent. Human interaction should be allowed.
FR:3	The system shall provide machine readable representations of data model objects.
Priority	Should have.
Description	External systems should be able to retrieve and parse resources without effort.
FR:4	The system shall allow for the data model to have geospatial properties.
Priority	Must have.
Description	Location is an important contextual attribute. It can apply for many different resources such as annotations or tracking the location of users.
FR:5	The system shall include a data model that is able to cluster resources.
Priority	Must have.
Description	Clustering of resources using a tag based (named bucket) or hierarchical approach allows them to be better managed.

Identifier: FR Name: System functional requirements (cont'd).	
FR:6	The system shall be able to do geometric operations.
Priority	Could have.
Description	Builds on FR:4. Geometric operations allows for measurements or tests on geospatial values.
FR:7	The system shall implement an event generator.
Priority	Should have.
Description	To allow greater extensability, the system should emit events to notify external actors that the system state has changed.

Identifier: NFR Name: System non-functional requirements.	
NFR:1	The system shall be deployable as a package.
Priority	Want to have.
Description	As far as it is possible, developers extending the system should not have to spend a substantial amount of time organizing, compiling or gathering dependencies. A packaged deployment should be seen as beneficial as it more easily integrates into the development or runtime environment.
NFR:2	The system shall be extended by experts.
Priority	Must have.
Description	Developers should be well versed in the system as not to break against the guidelines and requirements when/if extending it. Doing so could imply difficulties repurposing the extended system which should be avoided.
NFR:3	The system shall expose service interfaces that uses the HTTP protocol.
Priority	Should have.
Description	The use of restricted devices could imply that some application layer protocols of the Internet Protocol suite are excluded or considered time-costly to implement. HTTP should be considered to be highly interoperable.
NFR:4	The system shall use PostgreSQL for database operations.
Priority	Could have.
Description	PostgreSQL with the PostGIS extension enables geospatial objects which many functional requirements depend on.
NFR:5	Unit tests should exist for all data models.
Priority	Should have.
Description	Unit tests can partially serve as documentation of system usage and functionality.

Identifier: NFR Name: System non-functional requirements (cont'd).	
NFR:6	The data model should be designed to allow the inclusion of objects which cannot be described with a predefined schema.
Priority	Should have.
Description	It is not known beforehand if data model objects always can be defined with a prespecified set of attributes (schema). The data model should therefore include an allowance of emergent data that can be stored and managed at least at the application level but preferably at the lowest data level.
NFR:7	The system shall be implemented using a web application framework.
Priority	Should have.
Description	Web application frameworks provide essential and sufficient abstractions to allow rapid development of features.

Appendix C : Data model source code.

```
from django.db import models from django.contrib.gis.db import models
as gis_models from django.contrib.contenttypes.models import
ContentType from django.contrib.contenttypes import generic from
django.db.models.query import QuerySet import tagging

class Topic(models.Model):
    import datetime
    label = models.CharField(max_length=20)
    association = models.ForeignKey('self', null=True, blank=True,
related_name='child_topics')
    created_on =
models.DateTimeField(default=datetime.datetime.today)

    def __unicode__(self):
        return self.label tagging.register(Topic)

class Entry(models.Model):
    import datetime
    created_on =
models.DateTimeField(default=datetime.datetime.today)
    association = models.ForeignKey(Topic)

    def __unicode__(self):
        import time
        return "Entry created on %s" % time.strftime("%Y-%m-%d
%H:%M:%S", self.created_on.timetuple())

tagging.register(Entry)

class GeoData(gis_models.Model):
    content_type = models.ForeignKey(ContentType)
    object_id = models.PositiveIntegerField()
    association = generic.GenericForeignKey('content_type',
'object_id')
    xy = gis_models.PointField(srid=4326)
    altitude = models.FloatField()
    time_of_fix= models.TimeField()
    objects = gis_models.GeoManager()

    @staticmethod
    def from_x_y_z(x, y, z, time_of_fix):
        o = GeoData(xy="POINT(%s %s)" % (x, y),
time_of_fix=time_of_fix)
        o.altitude = z
        return o
    @staticmethod
    def from_GPGGA_NMEA(sentance):
        import time
        result = dict()
        (format,
        utc,
```

```

latitude,
northsouth,
longitude,
eastwest,
quality,
number_of_satellites_in_use,
horizontal_dilution,
altitude,
above_sea_unit,
geoidal_separation,
geoidal_separation_unit,
data_age,
diff_ref_stationID) = sentence.split(",")

latitude_in=float(latitude)
longitude_in=float(longitude)
if northsouth == 'S':
    latitude_in = -latitude_in
if eastwest == 'W':
    longitude_in = -longitude_in

latitude_degrees = int(latitude_in/100)
latitude_minutes = latitude_in - latitude_degrees*100
longitude_degrees = int(longitude_in/100)
longitude_minutes = longitude_in - longitude_degrees*100
result['latitude'] = latitude_degrees +
(latitude_minutes/60)
result['longitude'] = longitude_degrees +
(longitude_minutes/60)
result['time_of_fix'] = time.strftime("%H:%M:%S",
time.strptime(utc.split(".")[0], "%H%M%S"))
result['altitude'] = float(altitude)
return GeoData(xy="POINT(%s %s)" % (result['longitude'],
result['latitude']),
altitude =
result['altitude'],
time_of_fix =
result['time_of_fix'])

def isEast(self):
    return self.xy.x >= 0
def isNorth(self):
    return self.xy.y >= 0
tagging.register(GeoData)

def resolve_file_path(instance, filename):
    return "repository/%s/%s" % (instance.__class__.__name__,
filename)

class Media(models.Model):
    binary = models.FileField(upload_to="repository/Media")
    mimetype = models.CharField(max_length=64, editable=False,
default="application/octet-stream")

```

```

        association = models.ForeignKey(Entry)

tagging.register(Media)

class PredicateValueManager(models.Manager):
    def __init__(self, namespace, predicate):
        self.namespace = namespace
        self.predicate = predicate
        super(PredicateValueManager, self).__init__()

    def get_query_set(self):
        ns = URI.objects.get(value=self.namespace)
        p = URI.objects.get(value=self.predicate)
        return super(PredicateValueManager, self).get_query_set() \
            .filter(namespace=ns) \
            .filter(predicate=p)

class URI(models.Model):
    value = models.TextField()

    def save(self, force_insert=False, force_update=False):
        import lazr.uri
        self.value = unicode(lazr.uri.URI(self.value))
        super(URI, self).save(force_insert, force_update)

tagging.register(URI)

from karamel.core.fields import PickledObjectField
class PredicateValue(models.Model):
    association = models.ForeignKey(to=Entry)
    namespace = models.ForeignKey(URI,
related_name="predicatevalue_set_as_namespace")
    predicate = models.ForeignKey(URI,
related_name="predicatevalue_set_as_predicate")
    value = PickledObjectField() tagging.register(PredicateValue)

tagging.register(PredicateValue)

```

Appendix D : Unit test execution.

```
magnus@dev2:~/django-projects/karamel$ ./manage.py test core
Creation of spatial database test_karamel successful.
Creating table auth_permission
Creating table auth_group
Creating table auth_user
Creating table auth_message
Creating table django_content_type
Creating table django_session
Creating table django_site
Creating table django_admin_log
Creating table tagging_tag
Creating table tagging_taggeditem
Creating table core_topic
Creating table core_entry
Creating table core_geodata
Creating table core_media
Creating table core_uri
Creating table core_predicatevalue
Creating table gem_client
Creating table worldbordersextension_worldborders
Creating trigger functions for core_topic
Creating trigger functions for core_entry
Creating trigger functions for core_geodata
Creating trigger functions for core_predicatevalue
Creating trigger functions for core_media
Installing custom SQL for core.GeoData model
Installing custom SQL for gem.Client model
Installing custom SQL for worldbordersextension.WorldBorders model
Installing index for auth.Permission model
Installing index for auth.Message model
Installing index for admin.LogEntry model
Installing index for tagging.TaggedItem model
Installing index for core.Topic model
Installing index for core.Entry model
Installing index for core.GeoData model
Installing index for core.Media model
Installing index for core.PredicateValue model
.....Start worker
Resuming tests
Waiting for datamatrixworker
.
-----
Ran 15 tests in 26.015s

OK
Destroying test database...
```

Appendix E : Unit test source code.

```
import os
import time
from ipdb import set_trace

from django.test import TestCase, TransactionTestCase
from django.contrib.gis.db import models
from django.contrib.gis.utils import LayerMapping

from tagging.models import Tag, TaggedItem

from karamel.core.models import *
from karamel.worldbordersextension.models import *

def random_string():
    import string, random
    return "".join(random.sample(string.letters+string.digits, 8))

'''
    Data model testcases
    -----
'''

class TopicTest(TestCase):
    def setUp(self):
        self.maintopicname = "Test topic"
        self.childtopicname = "Child topic"

    def testSimpleAssertion(self):
        Topic.objects.create(label=self.maintopicname)
        self.assertEqual(len(Topic.objects.all()), 1)

    def testCreateSubtopic(self):
        Topic.objects.create(association=Topic.objects.create(label=self.maintopicname),
                               label=self.childtopicname)

        maintopic =
Topic.objects.get(label__exact=self.maintopicname)
        self.assertTrue(len(maintopic.child_topics.all()) == 1)
        self.assertEqual(maintopic.child_topics.all()[0].label,
self.childtopicname)

class EntryTest(TestCase):
    def setUp(self):
        self.topic = Topic.objects.create(label="Test topic")

    def testSimpleAssertion(self):
        Entry.objects.create(association=self.topic)
        self.assertEqual(len(Entry.objects.all()), 1)

class MediaTest(TestCase):
    def setUp(self):
```

```

        self.topic = Topic.objects.create(label="Test topic")
        self.entry = Entry.objects.create(association=self.topic)
        self.temporary_filename = random_string()+".txt"
        self.media_object_list = []

    def testCreateWithoutMimetype(self):
        import os
        from django.core.files.base import ContentFile

        media = Media(association=self.entry)
        media.binary.save(self.temporary_filename, ContentFile('---
'))
        self.assertEqual(media.mimetype, "application/octet-stream")

        filepath = media.binary.file.name
        self.assertTrue(os.path.exists(filepath))
        self.media_object_list.append(media)

    def testCreateWithMimetype(self):
        import os
        from django.core.files.base import ContentFile

        media = Media(association=self.entry, mimetype='text/plain')
        media.binary.save(self.temporary_filename, ContentFile('---
'))

        self.assertEqual(media.mimetype, "text/plain")

        filepath = media.binary.file.name
        self.assertTrue(os.path.exists(filepath))
        self.media_object_list.append(media)

    def tearDown(self):
        [media.delete() for media in self.media_object_list]

class GeoDataTest(TestCase):
    def setUp(self):
        self.topic = Topic.objects.create(label="Test topic")
        self.entry = Entry.objects.create(association=self.topic)

    def testGeoData(self):
        geodata = GeoData.objects.create(xy="POINT(0.0 0.0)",
                                         altitude=0.0,
                                         time_of_fix="12:00:00.0",
                                         association=self.entry)

        entryobject = Entry.objects.all()[0]

        entry_type = ContentType.objects.get_for_model(Entry)
        l = GeoData.objects.filter(content_type__pk=entry_type.id,
                                   object_id=entryobject.id)

        self.assertTrue(len(l) == 1)

```

```

self.assertTrue(l[0].isEast())
self.assertTrue(l[0].isNorth())

self.assertTrue(isinstance(l[0], GeoData))

def testGeoDataWithGPGGANMEA(self):
    geodata =
GeoData.from_GPGGA_NMEA("$GPGGA,114533.203,5651.0,N,01449.50,E,0,03,,0
.0,M,34.7,M,,0000*41")

    self.assertAlmostEqual(geodata.xy.x, 14.825)
    self.assertAlmostEqual(geodata.xy.y, 56.85)
    self.assertEqual(geodata.altitude, 0.0)
    self.assertTrue(geodata.isNorth())
    self.assertTrue(geodata.isEast())

def testGeoDataWithXYZ(self):
    geodata = GeoData.from_x_y_z(14.825, 56.85, 0.0,
"12:00:00.0")

    self.assertAlmostEqual(geodata.xy.x, 14.825)
    self.assertAlmostEqual(geodata.xy.y, 56.85)
    self.assertEqual(geodata.altitude, 0.0)
    self.assertTrue(geodata.isNorth)
    self.assertTrue(geodata.isEast)

class PredicateValueTest(TestCase):
    def setUp(self):
        self.topic = Topic.objects.create(label="Test topic")
        self.entry = Entry.objects.create(association=self.topic)

    def testPredicateValue(self):
        ns = URI.objects.create(value="http://testns.tld/")
        p =
URI.objects.create(value="http://testns.tld/p/temperature")
        pvoc = PredicateValue.objects.create

        pvoc(namespace=ns,
            predicate=p,
            value="10.0",
            association=self.entry)

        pvoc(namespace=ns,
            predicate=p,
            value="20.0",
            association=self.entry)

        pvoc(namespace=ns,
            predicate=p,
            value="30.0",
            association=self.entry)

        pvoc(namespace=ns,

```

```

        predicate=p,
        value="40.0",
        association=self.entry)

    entryobject = Entry.objects.all()[0]

    self.assertTrue(len(entryobject.predicatevalue_set.all()) ==
4)

class TagTest(TestCase):
    def setUp(self):
        self.topic = Topic.objects.create(label="Test topic")
        self.entry1 = Entry.objects.create(association=self.topic)
        self.entry2 = Entry.objects.create(association=self.topic)

    def testSimpleAssertion(self):
        Tag.objects.update_tags(self.entry1, 'entry1tag commontag')
        Tag.objects.update_tags(self.entry2, 'entry2tag commontag')
        Tag.objects.update_tags(self.topic, 'topictag commontag')

        l = list(Tag.objects.get(name='commontag').items.all())

        for ti in l:
            self.assertTrue(isinstance(ti.object, Entry) or
isinstance(ti.object, Topic))
            self.assertFalse(isinstance(ti.object, Media))

    def testAssociation(self):
        Tag.objects.update_tags(self.entry1, 'entry1tag commontag')
        Tag.objects.update_tags(self.topic, 'topictag commontag')

        l = [o.object for o in
TaggedItem.objects.filter(tag=Tag.objects.get(name='commontag'))]
        self.assertTrue(self.entry1 in l and self.topic in l)

'''
    Extension testing
    -----
'''

'''
    Proxy model test
'''

# Proxy model for metadata related to temperature via
PredicateValueManager
class TemperatureMetadata(PredicateValue):
    objects = PredicateValueManager(namespace='http://testns.tld/',
predicate='http://testns.tld/p/temperature')

class Meta:

```

```

    proxy = True

def as_celsius_to_fahrenheit(self):
    return 9.0/5.0 * self.value + 32

def as_fahrenheit_to_celsius(self):
    return 5.0/9.0 * (self.value - 32)

class ProxyModelTest(TestCase):
    def setUp(self):
        self.topic = Topic.objects.create(label="Test topic")
        self.entry = Entry.objects.create(association=self.topic)

        pvoc = PredicateValue.objects.create
        ns = URI.objects.get_or_create(value="http://testns.tld/")[0]
        p =
URI.objects.get_or_create(value="http://testns.tld/p/temperature")[0]

        pvoc(namespace = ns,
            predicate = p,
            value=10.0,
            association=self.entry)

        pvoc(namespace = ns,
            predicate = p,
            value=20.0,
            association=self.entry)

        pvoc(namespace = ns,
            predicate = p,
            value=30.0,
            association=self.entry)

    def testSimpleAssertion(self):
        # Assert that casting works
        o =
TemperatureMetadata.objects.filter(association=self.entry)[0]
        self.assertTrue(isinstance(o.value, float))
        o.value = 20.0

        # Assert that data isn't implicitly persisted
        o2 =
TemperatureMetadata.objects.filter(association=self.entry)[0]
        self.assertTrue(isinstance(o2.value, float))
        self.assertNotEquals(o2.value, 20.0)

        # Explicit save
        o.save()

        # Assert that data has been persisted
        o2 =
TemperatureMetadata.objects.filter(association=self.entry)[0]

```

```

        self.assertTrue(isinstance(o2.value, float))
        self.assertEqual(o2.value, 20.0)

    def testProxyExtension(self):
        #o =
TemperatureMetadata.objects.filter(association=self.entry)[0]
        o =
TemperatureMetadata.objects.filter(association=self.entry)[0]

        f = o.as_celsius_to_fahrenheit()
        c = o.as_fahrenheit_to_celsius()
        self.assertAlmostEqual(c, -12.222,3)
        self.assertEqual(f, 50.0)

'''
    GIS tests
'''

class GISTest(TestCase):
    def setUp(self):
        world_mapping = {
            'fips' : 'FIPS',
            'iso2' : 'ISO2',
            'iso3' : 'ISO3',
            'un' : 'UN',
            'name' : 'NAME',
            'area' : 'AREA',
            'pop2005' : 'POP2005',
            'region' : 'REGION',
            'subregion' : 'SUBREGION',
            'lon' : 'LON',
            'lat' : 'LAT',
            'mpoly' : 'MULTIPOLYGON',
        }

        world_shp =
os.path.abspath(os.path.join(os.path.dirname(__file__),
'../gis_test_resources/TM_WORLD_BORDERS-0.3.shp'))

        lm = LayerMapping(WorldBorders,
                           world_shp,
                           world_mapping,
                           transform=False,
                           encoding='iso-8859-1')

        lm.save(strict=True, verbose=False)

        topic = Topic.objects.create(label="Test topic")
        entry_sweden = Entry.objects.create(association=topic)
        entry_norway = Entry.objects.create(association=topic)
        entry_finland = Entry.objects.create(association=topic)

```

```

goc = GeoData.objects.create
self.gdo0 = goc(xy="POINT(14.941406 59.905100)",
                altitude=0.0,
                time_of_fix="12:00:00.0",
                association=entry_sweden)

self.gdo1 = goc(xy="POINT(8.393555 60.645903)",
                altitude=0.0,
                time_of_fix="12:00:00.0",
                association=entry_norway)

self.gdo2 = goc(xy="POINT(26.015625 62.989837)",
                altitude=0.0,
                time_of_fix="12:00:00.0",
                association=entry_finland)

def testPointInPolygon(self):
    entry_type = ContentType.objects.get_for_model(Entry)

    l = [self.gdo0, self.gdo1, self.gdo2]

    for entry in Entry.objects.all():
        gdo = GeoData.objects.get(content_type__pk=entry_type.id,
                                  object_id=entry.id)
        wbo = WorldBorders.objects.get(mpoly__intersects=gdo.xy)
        if wbo.name in ("Sweden", "Norway", "Finland"):
l.remove(gdo)

        self.assertEqual(l, [])

...
... Worker tests
...
class DataMatrixTest(TransactionTestCase):
    def setUp(self):
        self.topic = Topic.objects.create(label="Test topic")
        self.entry = Entry.objects.create(association=self.topic)
        self.media_object_list = []
        import time
        print "Start worker"
        time.sleep(5)
        print "Resuming tests"

    def testScan(self):
        import os
        from django.core.files.base import ContentFile

        media = Media(association=self.entry, mimetype="image/jpeg")
        media.binary.save('dm13.jpg',
ContentFile(open('/home/magnus/datamatrix/dm13.jpg').read()))
        self.media_object_list.append(media)
        print "Waiting for datamatrixworker"
        time.sleep(2)

```

```
    try:
        pv =
PredicateValue.objects.get(association=media.association)
        self.assertEqual(pv.value, '123456', "foo")
    except PredicateValue.DoesNotExist:
        self.fail("PredicateValue does not exist, was worker
running?")

def tearDown(self):
    [media.delete() for media in self.media_object_list]
```

Appendix F : PostgreSQL trigger generator.

```
from django.db.models.signals import post_syncdb
from django.db import connection, transaction
import karamel

called_post_syncdb_attach_functions = False
included_models = (
    karamel.core.models.Topic,
    karamel.core.models.Entry,
    karamel.core.models.GeoData,
    karamel.core.models.Media,
    karamel.core.models.PredicateValue
)

def post_syncdb_attach_functions(signal, **kwargs): #sender, app,
created_models, verbosity, interactive):
    global called_post_syncdb_attach_functions

    if not called_post_syncdb_attach_functions:
        cursor = connection.cursor()
        cursor.execute('''
                                CREATE LANGUAGE plpythonu;
                                CREATE OR REPLACE FUNCTION stompsend()
                                RETURNS trigger AS
                                $BODY$

try:
    import socket
    import stomper
    import simplejson as json

    c = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    c.connect((TD['args'][0], int(TD['args'][1])))
    fileobj = c.makefile('r', 0)

    responder = stomper.Engine()

    msg = stomper.connect('', '')
    fileobj.write(msg)

    send_message = stomper.send("/topic/karamel", json.dumps(TD))
    fileobj.write(send_message)
except:
    pass
$BODY$

                                LANGUAGE 'plpythonu' VOLATILE
                                COST 100;
                                ALTER FUNCTION stompsend() OWNER TO
karamel;
                                ''')

    for model in kwargs['created_models']:
        if model in included_models:
            db_table = model._meta.db_table
```

```

        cursor = connection.cursor()

        querystring = '''CREATE TRIGGER
notification_trigger_for_%s
                        AFTER INSERT OR UPDATE OR DELETE
                        ON %s
                        FOR EACH ROW
                        EXECUTE PROCEDURE
stompsend('localhost', 61613);
                        ''' % (db_table, db_table)
        print "Creating trigger functions for %s" % db_table
        cursor.execute(querystring)

        called_post_syncdb_attach_functions = True

post_syncdb.connect(post_syncdb_attach_functions)

```

Appendix G : datamatrixworker.py.

```
#!/usr/bin/env python
import time
import sys

from PIL import Image
from client import StompClientFactory
from pydmtx import DataMatrix
from twisted.internet import reactor
import simplejson as json

'''
    Set up Django environment
'''
sys.path.append('/home/magnus/django-projects')
from django.core.management import setup_environ
from django.conf import settings as django_settings
from django.core.signals import request_finished

from karamel import settings
from karamel.core.models import Media, PredicateValue

setup_environ(settings)
settings.DATABASE_NAME = 'test_karamel'

class XStomp(StompClientFactory):

    def recv_connected(self, msg):
        self.subscribe('/topic/karamel')

    def recv_message(self, msg):
        d = json.loads(msg['body'])

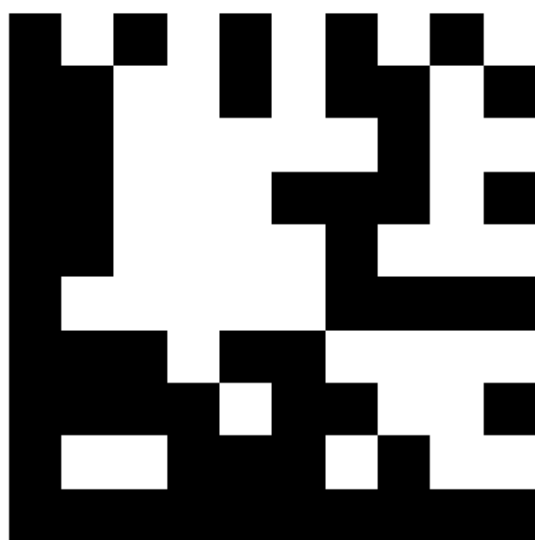
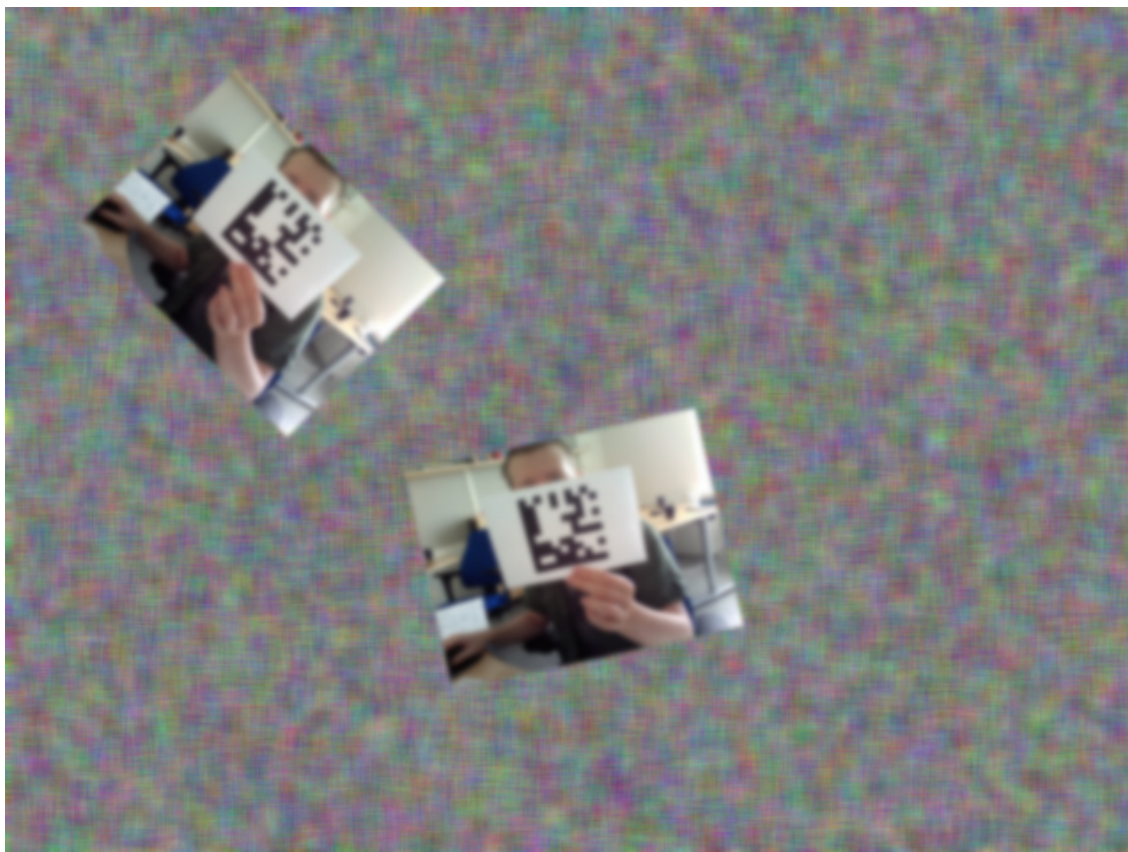
        if d['table_name']==Media._meta.db_table and \
            d['event']=="INSERT" and \
            d['new']['binary']!='' and \
            d['new']['mimetype'] == "image/jpeg":
            try:
                dm_read = DataMatrix( gap_size=10 )
                img = Image.open( "%s/%s" % (settings.MEDIA_ROOT,
d['new']['binary']) )
                dmcode = dm_read.decode( img.size[0], img.size[1],
buffer(img.tostring()) )
                if dmcode:
                    print "have dmcode"

                media = Media.objects.get(id=d['new']['id'])
                pvoc = PredicateValue.objects.create
                pvoc(namespace="http://testns.tld/",
                    predicate="http://testns.tld/datamatrix",
                    value=dmcode,
                    association=media.association)
            except:
```

```
        print "Exception"
    request_finished.send(sender=self) # Implicitly closes
database connection

reactor.connectTCP('localhost', 61613, XStomp())
reactor.run()
```

Appendix H : Data matrix image with source image.



123456



Växjö
University

Matematiska och systemtekniska institutionen
SE-351 95 Växjö

Tel. +46 (0)470 70 80 00, fax +46 (0)470 840 04
<http://www.vxu.se/msi/>