

Specifying the Global Execution Context of Computer-Mediated Tasks: A Visual Notation and a Supporting Tool

Demosthenes Akoumianakis

Department of Applied Informatics & Multimedia, School of Applied Technologies, Advanced Technological Education Institution of Crete, Crete, Greece.

Email: da@epp.teicrete.gr

Received August 4th, 2009; revised September 2nd, 2009; accepted September 14th, 2009.

ABSTRACT

This paper presents the notion of the global execution context of a task as a representational construct for analysing complexity in software evolution. Based on this notion a visual notation and a supporting tool are presented to support specification of a system's global execution context. A system's global execution context is conceived as an evolving network of use scenarios depicted by nodes and links designating semantic relationships between scenarios. A node represents either a base or a growth scenario. Directed links characterize the transition from one node to another by means of semantic scenario relationships. Each growth scenario is generated following a critique (or screening) of one or more base or reference scenarios. Subsequently, representative growth scenarios are compiled and consolidated in the global execution context graph. The paper describes the stages of this process, presents the tool designed to facilitate the construction of the global execution context graph and elaborates on recent practice and experience.

Keywords: *Non-Functional Requirements, Software Evolution Artifacts, Global Execution Context, Tools*

1. Introduction

Over the years a plethora of techniques have been developed to manage functional requirements of a software system. Some of these techniques focus on modeling and implementing functional requirements using constructs such as goals, scenarios, use cases, or notations such as UML diagrams and dedicated UML profiles. More recent efforts shift the focus to packaging and deploying functional requirements as reusable components and Web services for program-to-program interactions. In particular, service-oriented architectures (SOAs) appropriate the benefits of Web services to make it easier to exploit software assets from many types of components in sophisticated new solutions, without complex integration projects. Nevertheless, in all cases current thinking is dominated by concerns focusing on the lower levels of an enterprise infrastructure—how to create, manage and combine business services providing data and logic. These efforts and the supporting techniques to managing functional requirements are characteristic of the prevailing paradigm in software development, which can be broadly qualified as construction-oriented. At the core of this paradigm is the goal of designing what a software system is expected

to do, and to this end, the software design community has faced a variety of challenges in an effort to provide insights to the process of constructing reliable, robust and useful interactive systems and services.

The advent and wide proliferation of the Internet and the WWW have expanded an already over-populated software design research agenda, bringing to the surface the compelling need to account for a variety of non-functional requirements such as portability, accessibility, adaptability/adaptivity, security, scalability, ubiquity etc. Some of them are well known to the software engineering community, while others challenge established engineering methods and work practices. For instance, a long-standing premise of user-centered development is that of 'understanding users'; but users are no longer sharply identifiable, homogeneous or easily studied [1]. Furthermore, the tasks users carry out keep changing both in type and scope [2], with every new generation of technology, from desktop systems to mobile and wearable devices and the emergence of ubiquitous environments. The radical pace of these technical changes and the proliferation of myriad of network attachable devices introduce novel contexts of use, requiring insights, which are frequently beyond the grasp of

software designers.

Recognition of these challenges has motivated recent calls for departing from construction-specific software design techniques towards evolution-oriented methods and tools. In this vein there have been proposals aiming to provide creative interpretation of best practices (e.g., by devising new modeling constructs [3,4], building dedicated UML profiles [5], specifying architectural pattern languages [6], etc.) in an effort to establish mechanisms for analyzing and/or abstracting from salient features of software artifacts. Despite recent progress, designing systems to cope with change and evolution remains a challenge and poses serious questions regarding the design processes needed, the appropriate methodology and the respective instruments. One research path aiming to establish the ground for such informed design practices concentrates on non-functional requirements (NFRs) as a means to shift the focus away from what a software system is expected to do towards how it should behave under specified conditions. NFRs or quality attributes represent global constraints that must be satisfied by the software. Such constraints include performance, fault-tolerance, availability, portability, scalability, abstraction, security and so on. Despite their recognition by the software engineering community, it is only recently (*i.e.*, in the early 90s) that researchers have embarked in efforts aiming to assess their relevance to and implications for software development [3,4,7]. Nevertheless, in contrast to functional requirements their non-functional counterparts have proven hard to cope with for a variety of reasons [8]. Firstly, most of them lack a standard connotation as they are being treated differently across engineering communities and software development disciplines (*i.e.*, the same or similar NFRs hold different meaning for say, platform developers and usability experts). Secondly, they are abstract, stated only informally and requiring substantial context-specific refinement to be accounted for. Thirdly, their frequently conflicting nature (e.g., scale of availability may conflict with performance) makes step-by-step implementation or verification of whether or not a specific NFR is satisfied by the final product, extremely difficult. These are some of the reasons why NFRs are not easily incorporated into standard software engineering tools and practice.

SOA provide a new context for revisiting several NFRs and their management during software development. Nevertheless, current efforts are almost exclusively concentrated on qualities such as abstraction, messaging, service discovery, data integration, security, service orchestration/composition, etc, to facilitate two key principles: 1) creation of business services with defined interfaces so that functionality can be built once and then consumed as required and 2) separation of the provision of the services from their consumption. In this endeavor, the software design community continues to devise ab-

stractions (*i.e.*, components, visual notation, models and tools) which make construction-oriented artifacts first-class objects, dismissing or undermining software evolution and the special value NFRs have in this context. On the other hand it is increasingly recognized that software evolution is steadily overtaking in importance software construction. Indeed, Finger [9] argues that ‘...the ability to change is now more important than the ability to create systems in the first place’.

An alternative approach to address this challenge may be grounded on establishing the appropriate level of abstractions to make evolution (rather than construction) artifacts explicit, traceable and manageable in the course of software development. This implies devising abstractions that allow us to expose how change is brought about, what it entails, how it is put into effect and how it may be traced and managed. To meet this goal, there are key milestones likely to catalyze future developments. Our understanding of this challenge leads to the conclusion that we need 1) modeling approaches directing analysis towards early identification of components that relate to the cause of change, the subject of change and the effect of change and 2) tools for effecting change in a compositional fashion, thus relating change to local components which are assembled without requiring global reconfiguration of the system. In this vein, the present work considers change management at a new level of abstraction by promoting a shift in the unit of analysis from task- or activity-level to task execution contexts. It is argued that managing change is synonymous to coping with complexity and entails a conscious effort towards designing for the global execution context of computer-mediated tasks. Our normative perspective is that software designers should increasingly be required to articulate the global execution context of a system’s tasks, rather than being solely concerned with the development of an abstract task model from which incrementally, either through mappings or transformations, a platform-aware version of the system is generated. Moreover, designing for the global execution context is a goal to be satisfied rather than fulfilled. To this end, a new method and a supporting tool is described which allow designers to reason proactively (*i.e.*, from early concept formation through to design, implementation and evaluation) about the global execution context of designated tasks. Both the method and the tool provide a step in the direction of making change a first-class design object accounted for explicitly by articulating the parameters likely to act as ‘drivers’ of change. This is facilitated by an analytical approach aiming to unfold, identify, represent and implement alternative computational embodiments of tasks suitable for a range of distinct task execution contexts considered relevant and appropriate.

The remainder of the paper is structured as follows. The next section considers change management in in-

formation systems and frames the problem in two theoretical strands relevant to this work, namely change as evolution and change as intertwining non-functional requirements. This contrast offers useful insight to some of the research challenges preventing the development of methods for effectively coping with changes. Then, the paper elaborates on and defines the notion of a system's global execution context, which forms an abstraction for addressing complexity in software evolution rather than software construction. The following section describes the i-GeC tool, which allows incremental specification of the global execution context by using scenarios. Our reference example is a light ftp application initially designed for desktop use. The paper is concluded with a discussion on the contributions of this work and a brief note on implications and future work.

2. Motivation and Related Work

Change in interactive software is inherently linked with the context in which a task is executed. Typical context parameters include the target user, the platform providing the computational host for the task and/or the physical or social context in which the task is executed. Each may give rise to a multitude of potential drivers for change. Therefore, it stands to argue that change management is about coping with complexity in construction as well as in evolution. Managing complexity in construction has been coined with the handling of functionality. Specifically, through the history of software design the primary focus has been on accommodating functional requirements so as to develop systems that meet specific user goals. The resulting systems could cope with minimal and isolated changes, related primarily to the user, since no other part of the system's execution context (*i.e.*, platform or context of use) was conceived as viable to change. On the other hand, complexity in evolution is a more recent challenge attributed to the adoption of the Internet and the proliferation of Internet technologies and protocols. These developments have brought about an increasing recognition of the catalytic role of NFRs and have necessitated a paradigm shift in the design of interactive software so as to explicitly account for quality attributes such as abstraction, openness and platform independence, interoperability, individualization, etc. Despite the fact that complexity in construction and complexity in evolution may seem as competing at first glance (*i.e.*, evolution frequently implies improvements in the functional requirements), our intention is to argue that they bring about complementary insights, which may prove beneficial to the development of systems which are easier to manage and use.

2.1 Change as Evolution

The term evolution generally refers to progressive change in the properties or characteristics of the subject of evo-

lution (*i.e.*, software). A common view to conceive software evolution is to focus on mechanisms and tools whereby progressive change in program characteristics (e.g., functionality) and growth in functional power may be achieved in systematic, planned and controlled manner [10]. This may be conceived from various perspectives and viewpoints. For instance, it may be viewed from the perspective of software engineering processes and thereby explain the proliferation of iterative software development models such as agile programming [11] and extreme programming [12], etc. It may also be related to evolution in requirements and requirements management [13], giving rise to methods for tracing evolving components [14,15], localizing changes to components and developing component interoperation graphs [16], framing change to scenarios and supporting scenario evolution [17,18], etc. Whatever the perspective adopted, it is widely accepted that the ability to change is now more important than the ability to create systems in the first place [9]. Change management becomes a first-class design goal and requires business and technology architecture whose components can be added, modified, replaced and reconfigured. The implication is that the complexity of software has definitely shifted from *construction* to *evolution*. As a result new methods and technologies are required to address this new level of complexity.

In the past, the software design community addressed complexity in construction by devising abstractions (*i.e.*, components, visual notation, models and tools) to make construction-oriented artifacts first-class objects of design. This paradigm has catalyzed developments and facilitated breakthroughs in areas such as: 1) data management (leading from the early conception of the relational model to more recent proposals [19]), 2) software design (progressively shifting from structured techniques to object orientation [20], the development of computer-aided software engineering tools [21,22], domain-specific design languages [23], architecture description languages [6] and software factories [24]), 3) user interface development (facilitating richer interactions with the advent of 2D and 3D graphical toolkits [25-28]), etc. The common theme in these developments is that complexity is addressed by establishing levels of abstraction, allowing software construction artifacts (expressed as models of some sort, code, or processes) to become first class objects. It can therefore be argued that the problem of complexity in software evolution amounts to establishing the appropriate level of abstraction to make evolution artifacts explicit, traceable and manageable. That is to say, we need to find the abstractions that allow us to expose how change is brought about, what it entails, how it is put into effect and how it may be traced and managed. As already stated, our understanding of this challenge leads to the conclusion that we need 1) modeling approaches directing analysis to the identifica-

tion of components that relate to the cause of change, the subject of change and the effect of change and 2) tools for effecting change in a compositional fashion, thus relating change to local components which are assembled without requiring global reconfiguration of the system.

2.2 Change and Non-Functional Requirements

Change as evolution of functional requirements is of course valid but it can only explain partially why modern information systems need to change. In fact, there is evidence to suggest that most of the changes in modern information systems do not concern functional components but their connections and interactions [9]. This explains recent efforts aiming to frame change in relation to NFRs [8] and architectural quality attributes [29,30] such as such as adaptability [31], portability [32], run-time adaptive behavior [2,33], etc. Through these efforts, it becomes increasingly evident that NFRs concern primarily environment builders rather than application programmers. Nevertheless, there is an equally strong line of research aiming to make NFRs visible and accountable for as early as possible in the software design lifecycle by developing conceptual models and dedicated notations. Specifically, there are techniques aiming to classify NFRs through taxonomies [34], develop representational notations for using NFRs [7], advance process-oriented instruments for working with them [7,29] and study their relationship to software architecture [30,35]. Although these techniques have evolved in separate engineering communities (each with its own point of view) and have typically been performed in isolation, they share common ground. For instance, they recognize the important role to be played by methodological concepts and supporting technology that promote architectural insight through suitable first-class objects. Phrased differently, software architecture quality attributes promote a gross decomposition of systems into *components* that perform basic computations and *connectors* that ensure that they interact in ways that make required global system properties to emerge. Thus establishing abstractions at the level of software architecture may help manage complexity in software evolution.

2.3 Framing Change to the Task's Execution Context

The present work focuses on treating change from the early stages of information systems development where a variety of critical decisions are taken regarding architecture, platforms, tools to be used, expected and foreseen behaviors. Our primary concern is to advance a proposal rooted in the anticipation of change and its incremental localization in components. To this end, we use the notion of task execution context to define a particular type of scenarios that are allowed to grow. Then the global

execution context of a task (or a piece of functionality) is an instance in a continuum of revisions and extensions of the designated task's execution context. Consequently, managing change amounts to a conscious effort towards designing for the global execution context of computer-mediated tasks.

The execution context of a task is understood in terms of a triad <Users, Devices, Context>. Users represent the end (target) users - individuals or communities of users - who experience an interactive artifact through which the task is carried out. Devices refer to the technological platform used to provide the computational embodiment of the interactive artifact. Finally, context is a reflection on the (physical and social) context of use in which the task is executed. It is worth noting that none of these relate to functional properties of the task. Then, designing for the global execution context of a particular task with specified functional requirements is directly related to unfolding the rationale for and the artifacts encountered in a range of plausible task execution contexts. Interpreting the above rather theoretical concept in terms of practical design guidelines raises several issues with two standing out very promptly. The first is the commitment towards exploring and managing complex design spaces, while the second is the shift of engineering practice towards abstract and specification-based techniques. Although neither is entirely new to the software design community (e.g., see the works by MacLean [36] on design space analysis, the work on DRL by [37], etc, as well as recent advances in device-independent mark-up languages such as UIML (<http://www.uiml.org/>) and model-based development tools such as Teresa [32]), their meaning and exploitation is slightly different in the context of the present work.

3. The Global Execution Context of Tasks

The premise of the present work is that software design lacks a coherent and detailed macro-level method – in the sense defined in [38] – for the management of change during the early stages of development where critical decisions on architecture, tools and platforms to be used, are taken. Consequently, our interest is in establishing an integrated frame of reference for identifying and propagating change (*i.e.*, new requirements or evolution in requirements) across stages in the course of the design and development processes so as to facilitate designing for the global execution context (GeC) of tasks.

3.1 Motivating Example & Terminology

It is useful to conceive the global execution context of a task as a space of transformations depicting possible and/or desirable mappings of a task's abstraction to alternative non-functional contexts (*i.e.*, interaction platforms, contexts of use or user profiles). **Figure 1** provides an illustrative example from the field of user inter-

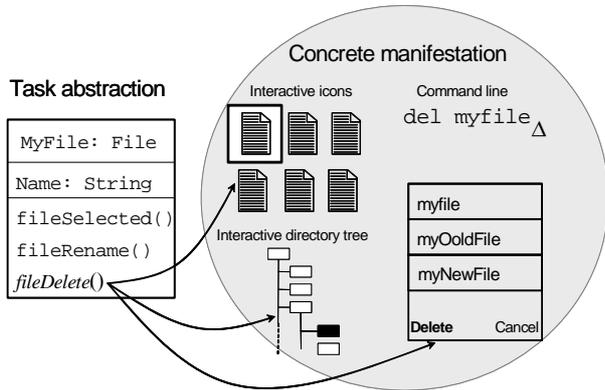


Figure 1. Possible transformations to derive the global execution context of a task

face engineering. Specifically, the figure presents schematically possible transformations for an abstract task ‘fileDelete’ of a hypothetical file management application to distinct concrete manifestations (potentially) suitable for different non-functional execution contexts. It is worth noticing that the example presents a case that challenges current conceptions of cross-platform or portable software in the sense that concrete manifestations need not be bound to native platform-specific elements; instead, they may use customized facilities, domain-specific and/or expanded components.

Being able to explicitly foresee and design a system so that it can cope with all possible changes in its execution context is probably a utopia, given the current state of the art in systems thinking and engineering. Nevertheless, if we delimit the qualification ‘all possible changes in the task’s execution context’ to all known, or foreseen and explicitly modeled changes (within the scope of a service-oriented architecture), then it is possible to define a context-sensitive processing function which under certain circumstances will deliver the maximally preferred transformation of an abstract task to a concrete instance [39]. Consequently, understanding and designing for the global execution context of a system’s task (*i.e.*, supporting the task’s execution across all designated non-functional contexts) entails some sort of mechanism or service for linking to, rather than directly calling, different implemented components complying to/supported by a designated service-oriented architecture. Equally important is to consider how the service-oriented architecture is to view and link to radically different execution contexts and platforms. In recent writings, both in research and development communities, this dimension is dismissed resulting in proposals for SOA that cannot cope with radically different non-functional execution contexts.

To provide further insight, let us abstract from the details of **Figure 1** to describe a more general situation where our abstract task T_2 (*i.e.*, delete a file) is assigned

to two distinct realizations as shown in **Figure 2**. The first, denoted with the solid line, refers to task execution on a desktop device, which requires that the selection list is presented (S_1), the user makes a choice (S_2) and subsequently the command is issued (S_3), followed by a confirmation dialogue (S_4). The second realization is using a mobile device. Once again the selection list is presented (S_1), but this time in order for the user to make a choice the system augments interaction initiating a scanning interface S_2' . Once the selection is made the command is issued (S_3) followed by a confirmation dialogue (S_4).

It is worth pointing out that despite the simplicity of the example, it poses several challenges. First of all, for any given task one can easily identify several additional realizations (execution contexts) depending on the platform or toolkit, the context of use and/or the target user. Thus, one issue is enumerating requirements and encoding alternatives, but also allowing for incremental updates and evolution to accommodate new realizations. Secondly, irrespective of the task’s execution context the functional requirement remains the same (*i.e.*, delete a file). The cause of change is therefore due to a designated set of NFRs. It may also be argued that prevalent NFRs such as portability or platform independence may not suffice to capture the essence implied by some of these changes. For instance, if scanning is implemented as a reusable interaction library, it signifies an *augmentation* of the target platform whereby the scanning functionality is introduced as new interaction technique assigned to designated interaction elements. This is totally different from a hard-coded implementation of the scanning interface to suit a specific interaction scenario or system. Similarly, one could envisage alternatives to augmentation such as platform *expansion* (*i.e.*, to increase the range of interaction elements of an existing platform) or *new platform development* (*i.e.* for a designated modality) and *integration* (*i.e.*, mixing components from different platforms). All these represent intertwining (and frequently conflicting) goals to inscribing non-functional qualities such as usability, portability, individualization, etc. Moreover, they are not intuitively associated with or assumed by prevalent NFRs. It stands to argue therefore that designing for the global execution context of a task

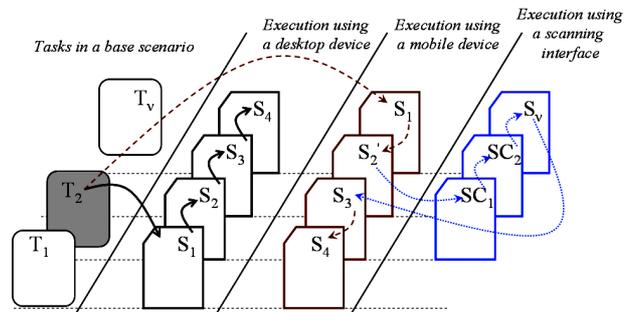


Figure 2. Tasks and execution contexts

entails an account of ‘hidden’ quality goals such as platform augmentation, expansion and integration, which are not so established in the software engineering literature. Furthermore, in many cases it is these ‘hidden’ quality goals that determine the type and range of non-functional contexts to be assigned to a task’s global execution context.

3.2 Modelling the Global Execution Context

Conceptually, the GeC of an abstract task T can be conceived as a five tuple relation $\langle T, g, S, f, C \rangle$ where g is the task’s goal to be achieved by alternative scenarios $s_i \in S$, and a context-sensitive processing function $f(s_i)$ which defines the maximally preferred instance of S , given a designated set of constraints C . Such a definition, allows us to model change in interactive software in terms of certain conditions or constraints which propagate alternative interactive behaviours to achieve task-oriented goals. Three types of constraints are relevant to the present work, namely user constraints, platform constraints and context constraints. User constraints are user-specific parameters designating alternative interaction and use patterns. Platform constraints relate to properties of a target device-specific execution environment. Context constraints designate external attributes of potential relevance to the task’s execution. Then, change δ in the execution context of a software system occurs if and only if there is at least one constraint in C whose parameter value has been modified so as to justify system transformation. The result of recognizing δ and putting it into effect causes the deactivation of $s_i \in S$, which was the status prior to recognizing δ and the activation of a new $s_j \in S$, which becomes the new status.

3.2.1 Basic Vocabulary and Notation

In our recent work, we have been developing a scenario-based approach in an attempt to formalize elements of the global execution context of computer-mediated tasks. In terms of basic vocabulary, the approach makes use of three constructs namely base (or reference) scenarios, growth scenarios and scenario relationships. Base scenarios depict situations in an existing system or a prototype, which are defined in terms of functionality. Growth scenarios extend reference scenarios in the sense that they describe new execution contexts for the functionality associated to the reference scenario.

Scenario relationships are used to capture semantic properties of the reference and growth scenarios. Two categories of relevant scenario relationships have been identified, namely those describing internal structure of scenarios in terms of components as well as those describing scenario realization. The former type of relationships is well documented in the literature (see [17]) and may be applied to any scenario independent of type (reference or growth). For the purposes of the present

work we have found two such relationships as being useful, namely subset-of and preference/indifference. Subset-of is the relationship defining containment between two scenarios. It declares that the functions of a scenario S_i are physically or logically part of another scenario S_j . S_i is termed the *subordinate scenario*, and S_j is termed the *superior scenario*. The subordinate scenario always encapsulates part of the action in the superior scenario. It should be noted that the subset-of relationship does not entail inclusion in the sense that execution of the superior scenario is suspended until the execution of the subordinate scenario is complete. Instead, it implies the set-theoretic notion of subset where the actions of the subordinate scenario are contained within or are the same as the set of actions of the superior scenario. Preference designates the existence of a preference order for two subordinate scenarios S_i and S_j of a superior scenario. Preference is specified by a preference condition or rule. When executed, the preference condition should place candidate subordinate scenarios in a preference ranking (indifference classes), while the most preferred scenario (first indifference class) is the one to be activated. The preference relationship is useful for specifying the context-sensitive processing function which activates/deactivates scenarios at run-time.

Scenario realization relationships provide details of the mapping (or transformation) between reference and growth scenarios and are intended to capture evolution of a reference scenario into growth scenarios. In general, two properties dictate the evolution of a base scenario into a growth scenario. The first relates to temporal aspects of growth scenario execution, while the second depicts the resources demanded for realizing the growth scenario. In terms of temporal aspects of execution these can be modeled either by alternative or parallel execution. The resources demanded can be modeled by relationships such as (platform) augmentation, (platform) expansion and (platform) integration. As the latter two are special cases of the alternative relationship, platform augmentation is the third scenario realization relationship used to complete the global execution context graph in the context of the present work. The alternative relationship links two scenarios when each serves exactly the same goals and one and only one can be active at any time. Alternative is the main operator for specifying adaptability of a system with regards to a designated quality attribute (e.g., platform independence). As already stated, two scenarios designated as alternative may be realized either by platform integration (typical case of multi-platform capability) or by platform expansion which assumes interoperability between the platform and another platform or third-party libraries used to expand the initial vocabulary of the platform. Moreover, two alternative scenarios are considered as indifferent with regards to all other quality attributes except the ones designated in the

alternative declaration (see preference). Augmentation captures the situation where one scenario in an indifference class is used to support or facilitate the mostly preferred (active) scenario within the same indifference class. For instance the scanning interface scenario in **Figure 2** augments file management when executed using a mobile device. In general, two scenarios related with an augmentation relationship serve precisely the same goal through different (but complementary) interaction means. Finally, parallelism refers to concurrent activation of scenarios serving the same goal. At any time two parallel scenarios preserve full temporal overlap. Parallelism may have two versions. The simplest version is when the scenarios utilize resources of the same platform. In this case the relationship is synonymous to concurrent execution of the scenarios (*i.e.* deleting a file using command line or an interactive directory tree) with full temporal overlap. The second version of parallelism is relevant when the scenarios utilize resources of different interaction platforms (toolkits). In this case, it is assumed that the two platforms are concurrently utilized and an abstract user interface can link with each one to make use of the respective interaction elements. This type of parallelism does not require interoperability between the platforms, as platform-specific interaction elements are not mixed. A typical example of this type of parallelism is when two users (*i.e.* a blind and a sighted user) are engaged in a collaborative application (*i.e.*, file management session) and the concrete user interface in each case utilizes interaction resources of different toolkits (one graphical toolkit for the sighted user's interface and one non-visual toolkit realizing the blind user's interface). This type of parallelism is not common in interactive applications, but when properly supported, it can serve a number of desirable features such as adaptivity to suit individualized requirements, concurrent modality-specific interaction as well as multimodality.

It should be noticed that the relationships discussed above are intended to serve the analysis of the global execution context as described earlier. All of them except the subset-of relationship are intended to address primarily non-functional qualities of scenarios. Consequently, these relationships are complementary to others proposed in the relevant literature (see for example [17]) for capturing semantic properties such as scenario complements, specialization, temporal suspension of a scenario until another scenario is completed (*i.e.* 'includes' relationship) or exceptional scenario execution paths (*i.e.* 'extends' relationship).

3.2.2 The Global Execution Context Graph

Collectively, the notational constructs described earlier are presented in **Table 1** and constitute the basic vocabulary of the global execution context notation (GeCn). Using this notation, designers can specify the requirements of the global execution context of a task as a graph.

This graph is typically a visual construction consisting of nodes representing scenarios and directed links representing scenario relationships. **Figure 3** illustrates an example of the global execution context graph of a task, namely select files, of a simple ftp application. This example will be further elaborated in the following section. The figure depicts, one reference scenario, namely 'Select files with desktop style' which through the containment operator links to 'Single file selection' and 'Multiple file selection'. Single and multiple file selection are parallel (*i.e.* weak notion of concurrency presented earlier, making use of resources of the same platform). The reference scenario as a whole (including the containments) is augmented with 'Select with scan on' which is a growth scenario containing two alternative options, namely 'One button/auto' and 'One button/manual' scanning. It is important to note that the designated growth scenarios do not represent change or evolution of the functional requirements of the application (either at the client or the server side). Instead, they designate a platform-specific non-functional requirement for supporting augmentation of interaction through scanning of interaction elements. On the other hand there is no pre-requisite as to how this augmentation is supported (*i.e.*, through programming or by augmenting toolkit libraries to facilitate scanning). This simple example suffices to make two claims regarding the global execution context graph of a task. Firstly, the technique is intended to represent 'hidden' requirements not commonly collected using conventional requirements engineering methods – thus it is complementary to rather than competing against such methods. Secondly, the technique is biased towards platform-oriented requirements leading to an improved insight on existing NFRs such as platform independence, portability, etc.

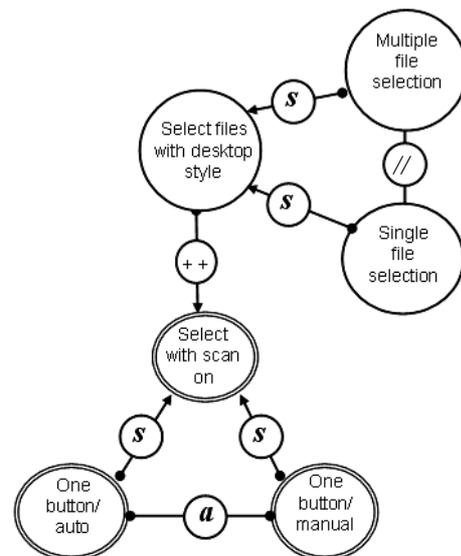


Figure 3. Example of a global execution context graph

Table 1. Basic notation

Symbol	Interpretation
(A)	Reference or base scenarios depict situations in an existing system or a prototype, which are defined in terms of functionality. They comprise at least one actor and one explicitly stated goal
(B)	Growth scenarios are always linked to a base scenario which they extend in the sense that they describe new execution contexts for the functionality associated to the reference scenario
— π —	'Preference' relationship designates the existence of a preference ranking between two or more scenarios; the preference ranking is conditional upon the preference rule (or condition)
— i —	'Indifference' relationship designates indifferent execution of two or more scenarios realized in the same design vocabulary (<i>i.e.</i> development platform)
— a —	'Alternative' relationship designates alternative realizations / embodiments of a scenario across distinct design vocabularies; at any time one and only one of the scenarios can be active
— ++ —	'Augmentation' relationship designates that a scenario is used to support or facilitate the mostly preferred (active) scenario within the same indifference class
— // —	'Parallel' relationship designates the concurrent activation of scenarios of a designated design vocabulary; at any time parallel scenarios preserve full temporal overlap
— ≈ —	'Interchangeable' execution designates parallel execution of scenarios in distinct design vocabularies; no requirement for interoperability as scenarios are not mixed
— s —	'Subset_of' defines containment of actions of one (subordinate) scenario into actions of another (superior) scenario; the subordinate scenario appears on the left hand side of the relationship

3.3 Stages in the Construction of Global Execution Context Graphs

Building the global execution context graph entails an iterative process of continuous refinement. It is both useful and important to be able to verify refinements of a task's global execution context graph so as to ensure consistency and correctness. To facilitate these tasks, a micro method and a supporting tool have been developed to provide guidelines for building the global execution context graph. The method and the tool serve two main goals, namely 1) encoding reference and growth scenarios in alternative representation forms and 2) incremental and evolutionary construction of the system's global execution context graph so as to allow incorporation of new requirements and requirements evolution (*i.e.*, versioning). **Figure 4** illustrates the conceptual stages involved in compiling the global execution context graph

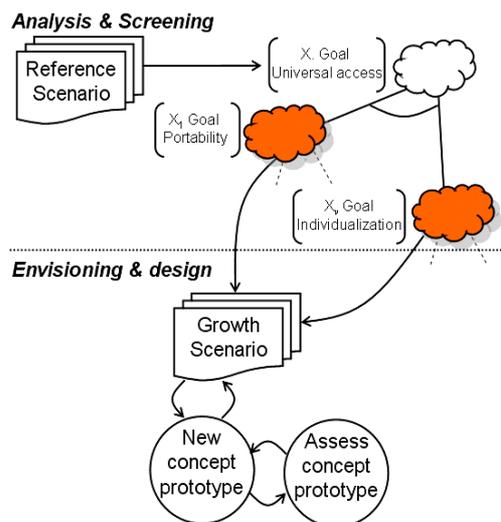


Figure 4. Process stages for building

using growth scenarios. As in the case of other scenario-based methods, it involves interplay between reflection (analysis and screening) and envisioning. The first step is usually a preparatory activity carried out by the analyst in collaboration with domain experts, and entails the formulation of reference scenarios. Once the scenario is formulated, typically in a narrative form, the screening process begins in an effort to compile the rationale for growth scenarios. To this end the choice of screening filters is important. One option is to screen the reference scenarios using designated NFRs so as to unfold breakdowns or deficiencies related to global system qualities (*i.e.* system architecture, platform commitment, interaction metaphor). In **Table 2** we provide an example of such NFRs-based screening of our reference ftp application.

Alternatively, screening may focus on other aspects of interactive software such as choice of interaction elements, dialogue styles, presentation, etc. In all cases, scenario screening assumes the availability of artifacts (e.g., narratives, pictures, user interface mock-ups, high fidelity prototypes, etc) and it entails a structured process whereby implicit or explicit assumptions embedded in an artifact (and related to the intended users, the platform used to implement the artifact and the context of use) are identified and documented. The essence of screening is in defining appropriate filters or adopting alternative perspectives to critique a base scenario. It is therefore a scenario inspection technique in the sense described in [40], which can be realized through different instruments.

Whatever choice of the screening instrument, it is imperative that screening should motivate growth scenarios so that the latter do not exist in vacuum. Instead, they should be related to the design breakdowns identified through screening and the designated new or evolving requirements. Consequently, the ultimate goal of growth scenarios is to capture evolution of requirements codified in base scenarios. Such evolution should depict new

Table 2. Screening using NFRs and corresponding design breakdowns

<i>NFR</i>		<i>Example of design breakdown</i>	<i>New or evolving requirement</i>
Platform independence		"... file transfer is not available as WWW or WAP application or service ..."	Allow choice of delivery medium or interaction style (i.e. HTML, WAP, Windows style)
Scalability		"... the system does not exhibit scalability to platform or access terminal capabilities ..."	Detect context of use and allow operation in text-only style through a kiosk
Adaptation	Adaptability	"...the system cannot be customized to diverse requirements..."	Support manual or automatic customization of interaction style (e.g., scanning)
	Adaptivity	"...when in operation the system does not monitor user's interactive behavior to adjust aspects of interaction..."	Provide auditory feedback upon completion of critical tasks to inform users on task completion state
Context awareness		"...the system takes no account of the context of use to modify its interactive behavior..."	Allow context monitoring and switching between designated interaction styles
Localization		"...the system can not be localized..."	Allow choice of language
Accessibility		"...the system is not accessible by certain target user groups..."	Interview user to determine motor, visual, cognitive capabilities and define adaptation

execution contexts for the tasks in the base scenario. In practice, growth scenarios result from relaxing the assumptions identified in the course of screening. Once agreed, growth scenarios may become more concrete through prototypes, which specify details of the new task execution contexts. It is important to note that our intention is to consider growth scenario management as an engineering activity [41] rather than a craft, and to contribute towards effective engineering practices for guiding the creation and refinement of scenarios. Consequently, our work links with recent proposals in scenario-based requirements engineering aiming to offer systematic scenario process guidance (see for example [42,43]) as well as key concepts and techniques of the Non-Functional Requirements Framework [8].

4. Designing for the Global Execution Context

To support designers in gaining insights and analysing the global execution context, a tool has been developed, namely interactive Global execution Context (i-GeC). i-GeC covers all three stages namely scenario recording, screening and growth scenario compilation. It does not however, embark into detailed design, which is beyond the scope of the present work. **Figure 5** depicts the logical view of i-GeC, summarising our notion of reference (or base) and growth scenarios as well as the scenario relationships relevant to this work. It should also be noted that the class model of **Figure 5** provides a scheme for interpreting the main components of the five tuple relation $\langle T, g, S, f, C \rangle$ used to conceptualise the GeC of a task. The only element not explicitly modelled is the context sensitive processing function f . However, this relates to the system's implementation and architectural model for processing (i.e. enabling/disabling) scenarios. As for the constraints they are assumed to be parameters of the class 'Artifact'. Another important consideration regarding the scheme of **Figure 5** is that, although there is a provision for goals, this should not be confused with functional requirements. In fact, this work is not concerned with this type of requirement. Instead, our interest is on non-functional

requirements and how they are translated into quality goals. As already mentioned earlier, some non-functional requirements (i.e. adaptability, portability and individualization) are well established in the relevant literature both in terms of scope and techniques used to cope with them (i.e. [3,8]). Others however are not so well established (i.e. toolkit augmentation, expansion, integration) but are considered very important to modelling the global execution context of a task. The latter type of non-functional requirements, partly motivate the work presented in this paper.

To illustrate the above, we will continue to make use of our ftp application allowing authorised users to connect to a server and subsequently manipulate local files (i.e. transfer, delete). For the purposes of our discussion, we will consider both the incorporation of new requirements and requirements evolution. A new requirement is to support ftp portability to a new platform (i.e., from desktop to PDA). As an example of requirements evolution we will consider various enhancements of the file selection task so as to support multiple selection by file category (i.e. select all files with an extension '.ppt') and selection through scanning on a PDA. Scanning is an interaction technique, which entails automatic manipulation of PDA interaction elements in a hierarchical fashion to reduce keystroke level actions. It is therefore conceived as a usability enhancement. Thus, in the remaining of this section, our aim is to show how from a given set of functional requirements we can progressively compile a specification of the system's new execution context supporting a PDA client with the enhanced file selection facilities.

4.1 Encoding/Recording Scenarios Using i-GeC

Encoding scenarios using a variety of media and representational tools is important, as it allows the designer to start with a high-level narrative description of a situation of use (see **Figure 6**, left hand side dialogue) and progressively transform it into a bulleted sequence, state diagram, use case model, etc., reflecting the designers' incremental improvement of understanding of the situa-

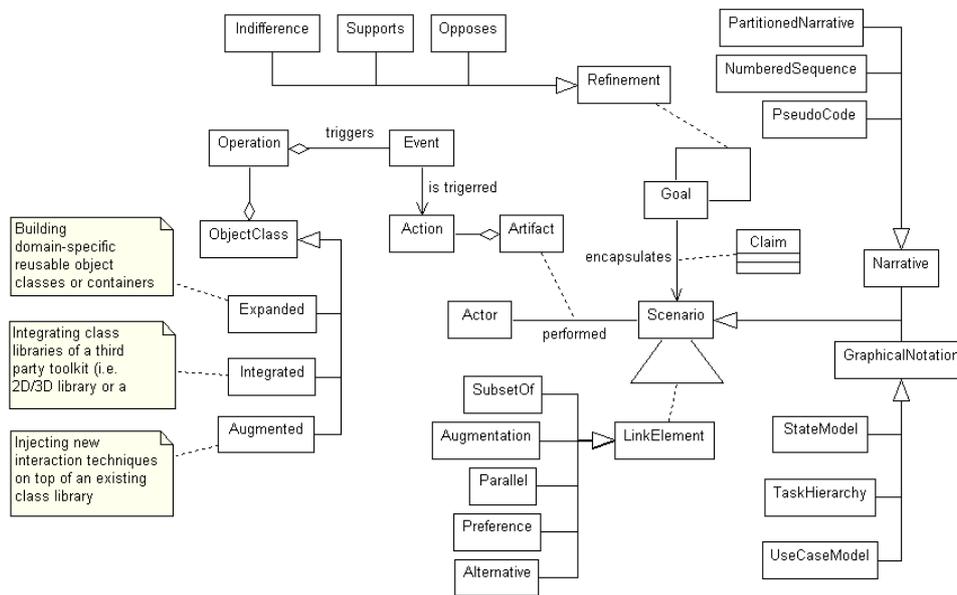


Figure 5. Class model of the global execution context of a task

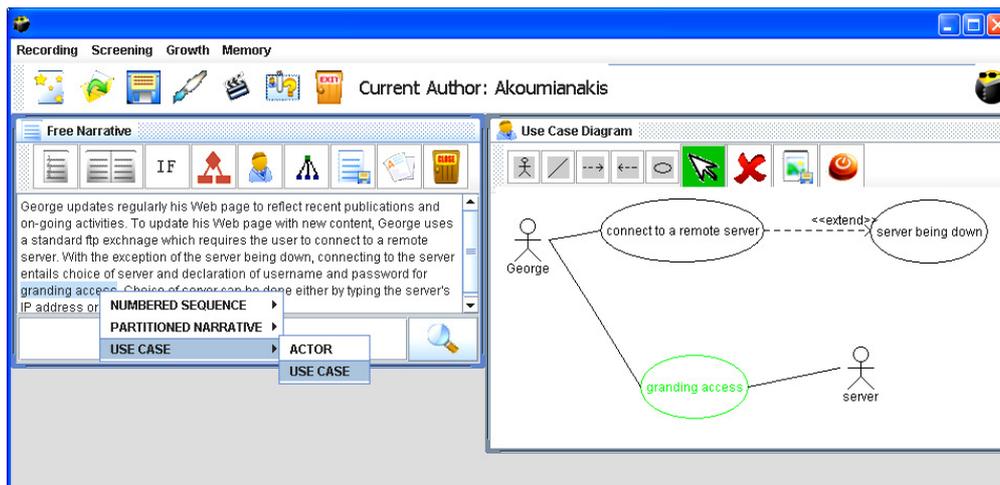


Figure 6. Encoding a reference scenario as use cases

tion. This transformation is user-driven in the sense that the user can employ simple cut & paste techniques or menu-driven dialogues to map textual elements in the narrative description to graphical elements in a specific visual notation (see for example **Figure 6** for a transformation of a narrative to a use case model). Each reference scenario can be incrementally refined. Reference scenario refinement involves detailed description of the scenario and compilation of more analytic views of the scenario codified as numbered sequence of activities, partitioned narrative, exception steps, state transitions, etc., as shown in **Figure 7**.

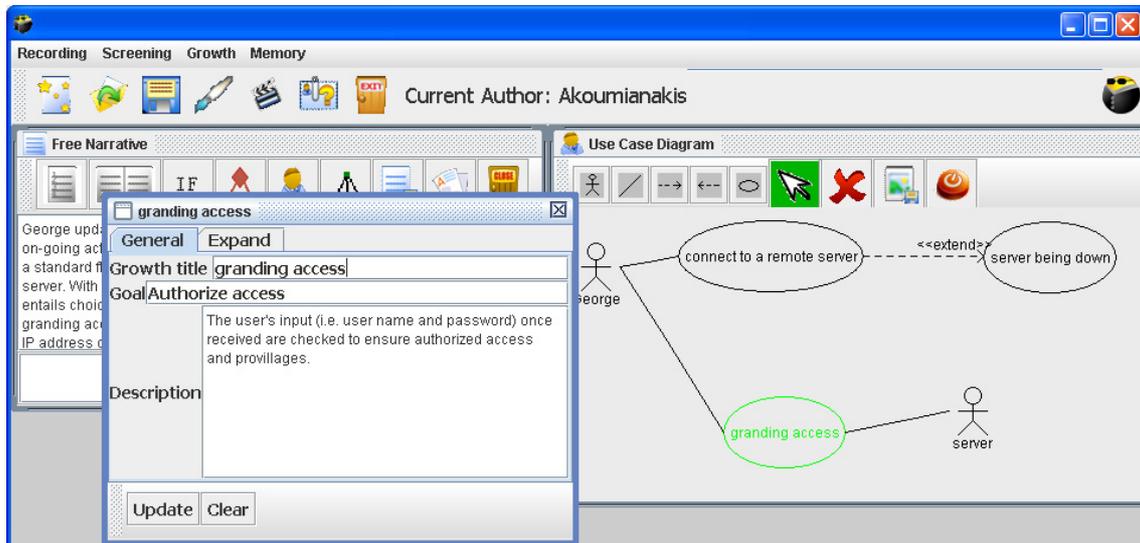
4.2 Scenario Screening with i-GeC

Following reference scenario recording, the screening stage seeks to provide a structured critique of the re-

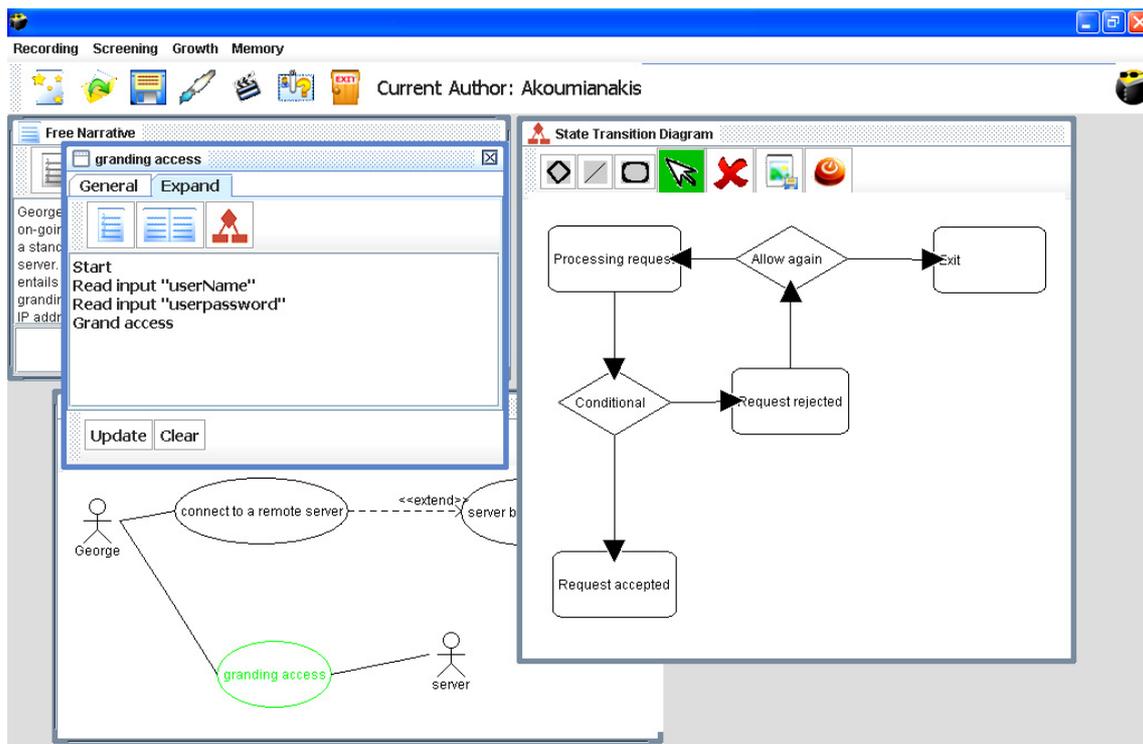
corded scenario so as to designate issues (in anticipation of change) or shortcomings. These shortcomings provide the rationale and the motives for subsequent compilation of growth scenarios (see next section). In the current version, screening a scenario follows the tradition of design space analysis using Questions Options & Criteria [36]. The analyst can designate both issues and options (potential solutions) as shown in **Figure 8**. All designated issues and options are codified per scenario and can be explored through the memory tool. This type of screening is intended only to record and make persistent the results of analysis.

4.3 Compiling Growth Scenarios and Building Global Execution Context Graphs with i-GeC

In i-GeC, the compilation of growth scenarios entails



(a)



(b)

Figure 7. Manipulation of reference scenarios. (a) Describing the reference scenario; (b) Expanding the reference scenario

reformulation of a use case type representation of the base scenarios. Specifically, to define a growth scenario, the user should first declare the growth case and then assign the appropriate relationships between the growth case and the base or other growth scenarios. **Figure 9** depicts an example where a growth scenario is introduced (**Figure 9(a)**) and subsequently elaborated (**Figure 9(b)**). In the example, ‘iPAQ connection’ is introduced as ‘alternative to’ the reference scenario ‘connect to a

remote server’ which represents functionality already supported by the ftp application. The growth scenario is motivated by the screening criterion of ‘user adaptability’ and the issue ‘how does the user type IP address’ (see **Figure 9(b)**). For the same growth scenario there may be more issues assigned. As shown, reference and growth scenarios are distinct elements represented as single-line and double-line ellipses respectively. The semantic scenario relationship is represented as an annotated link. The

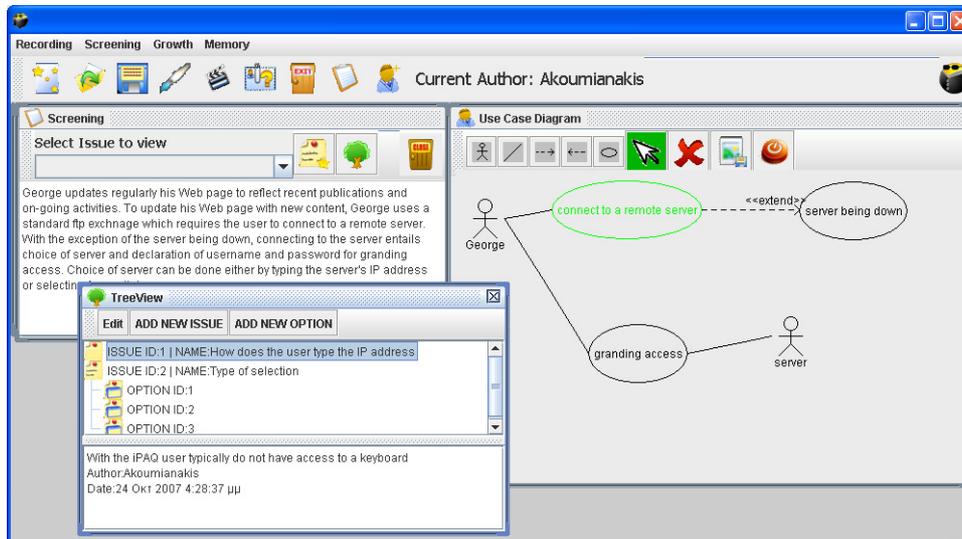
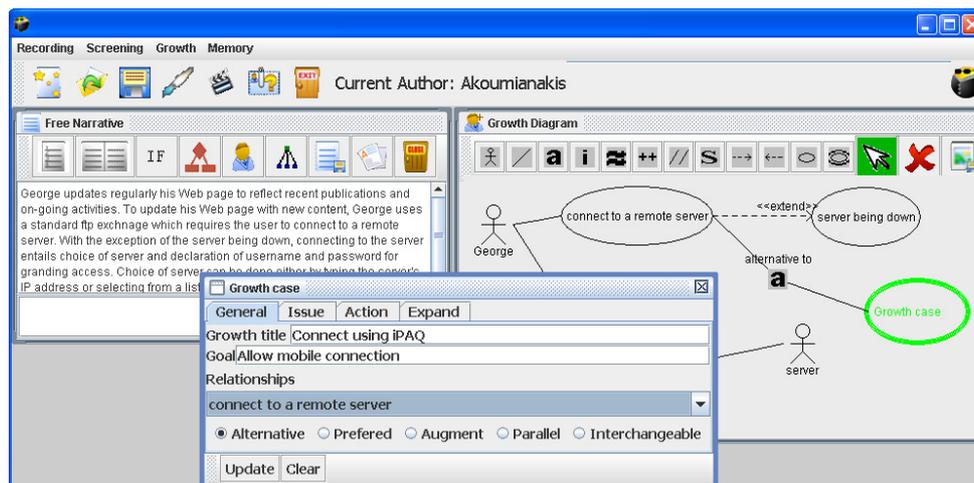
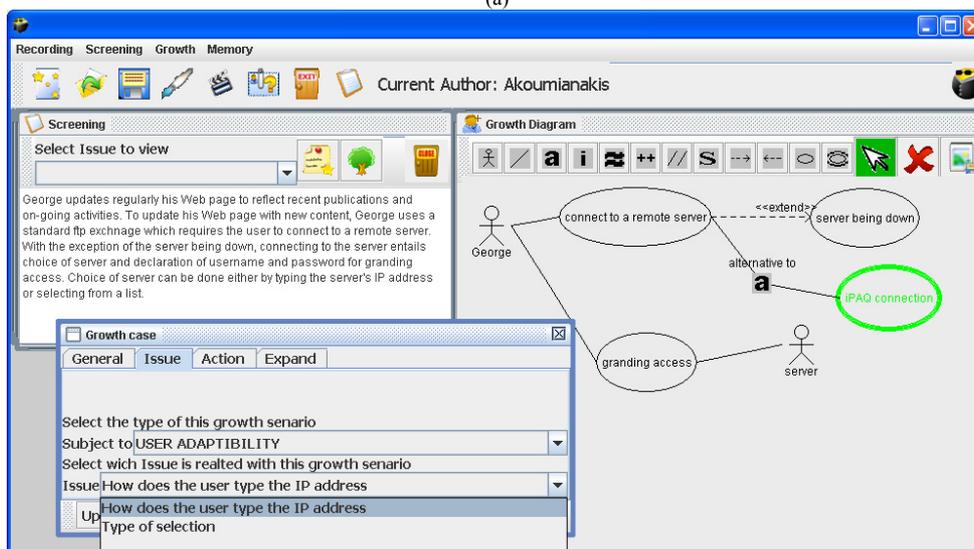


Figure 8. The screening stage



(a)



(b)

Figure 9. Populating a reference scenario. (a) Introducing a growth scenario; (b) Populating a growth scenario

growth scenario elaboration dialogue groups the properties of a growth scenario into four categories. The general properties of the growth scenario declare its name, description and relationship with other scenarios. In a similar fashion the user can assign the issues relevant to (or addressed by) the growth scenario, the pre- and post-conditions and supporting analysis (*i.e.* a state transition diagram, numbered sequence, partitioned narrative, etc). This provides a kind of verification for each growth scenario, since it ensures the minimum qualities required (*i.e.*, each scenario is assigned to a goal, each scenario is realized through a set of actions, etc). This issue is further elaborated later on in this paper.

Building the global execution context graph entails three steps: 1) devising growth scenarios; 2) assigning quality attributes to justify the derived growth diagram and 3) commenting on the pseudo verification applied to check the global execution context graph. **Table 3** provides a summary of the growth scenarios highlighting both the case of supporting ftp through PDA and the enhancement of the file selection task. These extensions are typically expressed as new/evolving requirements to be accommodated as growth cases of the initial base scenario. From the descriptions in **Table 3**, we can deduce that the global execution context of the new ftp application should include one additional growth scenario namely ‘Select with scanning’ and two parallel components designating that selection is augmented by two growth scenarios namely ‘One button/Auto’ or ‘One button/Manual’. This is depicted in **Figure 10**. The relationships between the various growth scenarios define the scale and scope of the system’s adaptable and/or adaptive behaviour. This offers useful insight to the range of anticipated changes and their implication on architectural abstraction, the choice of interaction techniques, as well as the conditions under which alternative styles of interaction are to be initiated.

At any time, designers can justify their decisions by rationalizing growth scenarios using non-functional quality models. Such models may be built in advance so as to establish global constraints on software design or in the course of building and rationalizing a task’s global execution context. **Figure 11** presents an example decomposition of the ‘accessibility’ quality in terms of alternatives or claims softgoals in the vocabulary of the NFR Framework [8]. Specifically, the model in **Figure 11** details that accessibility can be satisfied either by augmenting interaction through scanning, or by expanding a toolkit library with new interaction object classes or by integrating another toolkit class library. The relationships qualify the degree of satisficing a goal. Thus, augmentation and expansion support (*i.e.*, have a positive influence) on accessibility, while toolkit integration is indifferent. **Figure 11** on the left hand side represents the link between the non-functional quality model and the global

execution context graph. The rationale behind the combined model is intended to convey the following meaning: The iPAQ version of the ftp application should support a scanning interface which should allow selection in two alternative modes – one button with automatic scanning or one button with manual scanning of the highlighter.

Figure 12 depicts an interactive instance of the augmented ftp application with the scanning interface on and the multiple file selection facility (see **Table 3** and **Figure 10** for the rationale of the growth scenarios). As shown, scanning is activated through explicit function activation by pressing the button in the left bottom corner (see **Figure 12(a)**). Once activated the scanner gives focus in round-robin fashion to each control in a hierarchical fashion. It is worth noting the difference in the interactive behaviour for each object of focus. Thus, when scanning is activated and the object of focus is a text entry field the fill colour is changed (see **Figure 12(b)**) while when the object of focus is a button then the label is underlined (see **Figure 12(c)**). In **Figure 12(d)** the focus is on the Download Button (note the square around it). By pressing a hardware button on the device the scanner moves from one level to another (*i.e.* from scanning selection sets to scanning items within a selection set and vice versa). **Figure 12(e)** demonstrates the multiple file selection by checking files in a category. For example, when the “Remote Files” list has the focus, pressing the PowerPoint icon on the taskbar, all files with ppt extension in that list are selected. This multi-selection task adds checkboxes to the left of all items in the list, and files with ‘ppt’ extension are automatically checked. It should be noted that this type of selection is very useful as it reduces keystroke level interactions (*i.e.* avoids using the slider to locate files and multiple file checking), without changing the initial application in any other way.

4.4 Pseudo Verification of the Global Execution Context Graph

In its current version, the tool consolidates the global execution context graph in an XML document by implementing a pseudo-verification to ensure that the global execution context graph satisfies to some degree the criteria of completeness and redundancy. The tests performed aim to satisfy the following:

- Each scenario S_i in the global execution context graph $GeCg(S)$ where S denotes the reference system should be either a base scenario or a growth scenario
- Given a system S , for each base scenario $SB_i \in GeCg(S)$ there is at least one growth scenario $SG_j \in GeCg(S)$ related with SB_i
- Given a system S , then a growth scenario $SG_i \in GeCg(S)$ can be related with a base scenario $SB_i \in GeCg(S)$ or another growth scenario $SG_j \in GeCg(S)$;
- All scenarios are assigned to goals – informally, this

Table 3. Elaboration of growth scenarios

	Base scenario (Desktop)		Growth Scenario (PDA)		Growth Scenario (PDA & scanning)	
Initiator	Professional user		Professional user		User at home	
Context of use	The user outside the office		The user enters the classroom and wishes to ftp the file containing his slideshow		The user is at home and wishes to review his slideshow	
	<i>User</i>	<i>System</i>	<i>User</i>	<i>System</i>	<i>User</i>	<i>System</i>
Flow of events	<ul style="list-style-type: none"> o The user connects to the server and logs-in o The user carries out file selection o The user issues a command (by button press) 	<ul style="list-style-type: none"> o The system responds to notify user's login and presents the desktop embodiment of the user interface o The system notifies the user of current selection o The system executes the command o The system updates the display 	<ul style="list-style-type: none"> o The user connects to the server and logs-in o The user makes a selection from the list of patients o The user issues a command (using the light pen) 	<ul style="list-style-type: none"> o The system responds to notify user's login and presents the PDA embodiment of the user interface o The system list the currently selected file o The system executes the command 	<ul style="list-style-type: none"> o The user connects to the server and logs-in o The user declares selection mode o The user selects all files with .ppt extension 	<ul style="list-style-type: none"> o The system responds to notify user's login and presents the augmented PDA embodiment of the user interface o The system lists of files in a default style o The system initiates suitable style & informs the user o The system executes the command
Exceptions	<ul style="list-style-type: none"> ▸ Network problems ▸ Error in login procedure 		<ul style="list-style-type: none"> ▸ Network problems ▸ Error in login procedure 		<ul style="list-style-type: none"> ▸ Network problems ▸ Error in login procedure 	
Pre-conditions	<ul style="list-style-type: none"> ▸ User is authorized ▸ Desktop interface is available ▸ System has inferred the task's execution context resulting in HTML style being automatically initiated 		<ul style="list-style-type: none"> ▸ User is authorized ▸ User is in possession of the designated terminal ▸ System has inferred the task's execution context resulting in PDA style being automatically initiated 		<ul style="list-style-type: none"> ▸ User is authorized ▸ User is in possession of the designated terminal ▸ System has inferred the task's execution context resulting in PDA style being automatically initiated 	
Post-conditions	<ul style="list-style-type: none"> ▸ Designated files are successfully transferred 		<ul style="list-style-type: none"> ▸ Designated files are successfully transferred 		<ul style="list-style-type: none"> ▸ Designated files are successfully transferred 	
Relationships	<ul style="list-style-type: none"> ▸ Alternative to base scenario 		<ul style="list-style-type: none"> ▸ Alternative to base scenario 		<ul style="list-style-type: none"> ▸ Augments PDA style ▸ Parallel selection as separate growth scenarios 	

ensures that a scenario is devised to facilitate a designated goal of the system. Thus, there are no scenarios beyond the scope of the envisioned system;

- Each scenario can be satisfied by at least one goal – informally, the proposition aims to assert that each scenario is linked to at least one goal.

The above propositions are checked before the global execution context graph is transformed into XML. This allows a pseudo verification of the completeness, redundancy and understandability of a global execution context graph. Specifically, the propositions can be considered as necessary but not sufficient conditions for ensuring completeness. Clearly, as the global execution context graph is subject to refinement, no sufficient condition for com-

pleteness can hold. As for redundancy, the propositions aim to support a weak notion of redundancy, which asserts that no scenarios are included that would not be designated to goals. Obviously, the global execution context graph could incorporate redundancy both at the level of growth scenarios (*i.e.* alternative growth scenarios may exist which satisfy the same goal) and at the level of actions (*i.e.* alternative action sets may be employed to satisfy a user goal). Finally, regarding understandability, the propositions aim to ensure that all scenarios included in the global execution context graph are understandable by tracing their designated goals, which are considered valid. On the other hand, the propositions offer no guarantee that the global execution context graph can be understood.

At any instance, the GeCg can be traversed by selecting and following a particular path from start to end and understanding the system's behaviour under certain conditions. Path differentiation is always associated with a scenario relationship of type alternative or augments. Referring to our example, we can define possible traversals of the global execution context graph, differentiated by colour. Activating, the reference scenario results in an iPAQ embodiment of the designated task, with sin-

gle and multiple selection. Activation of scanning would augment the file selection process with scanning. As for the type of scanning, two alternative manifestations are available with only one being active at any point in time. In terms of system implementation requirements, it is important to note that the underlying intention is that both paths should co-exist, while through context-sensitive processing the system should decide on the choice of optimal path.

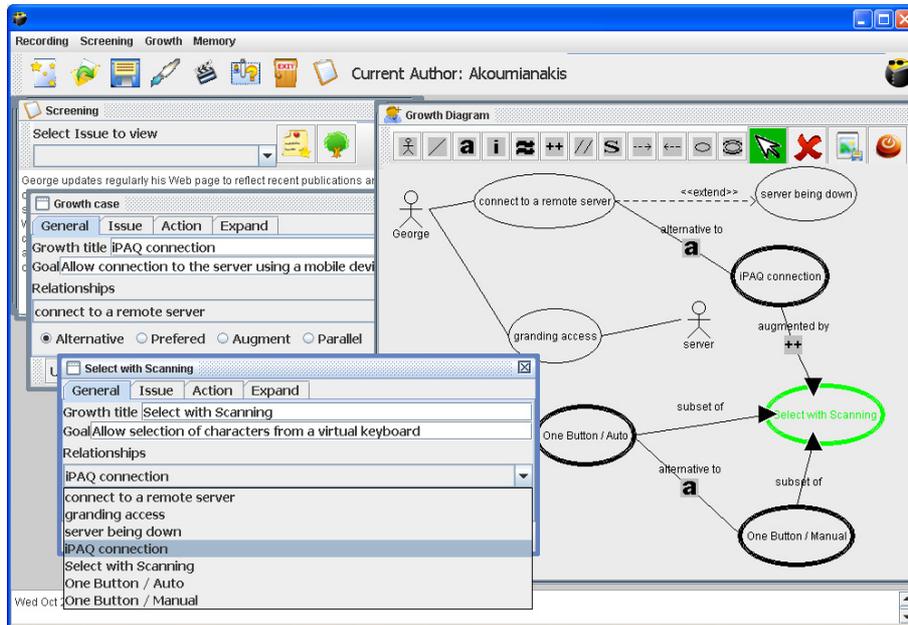
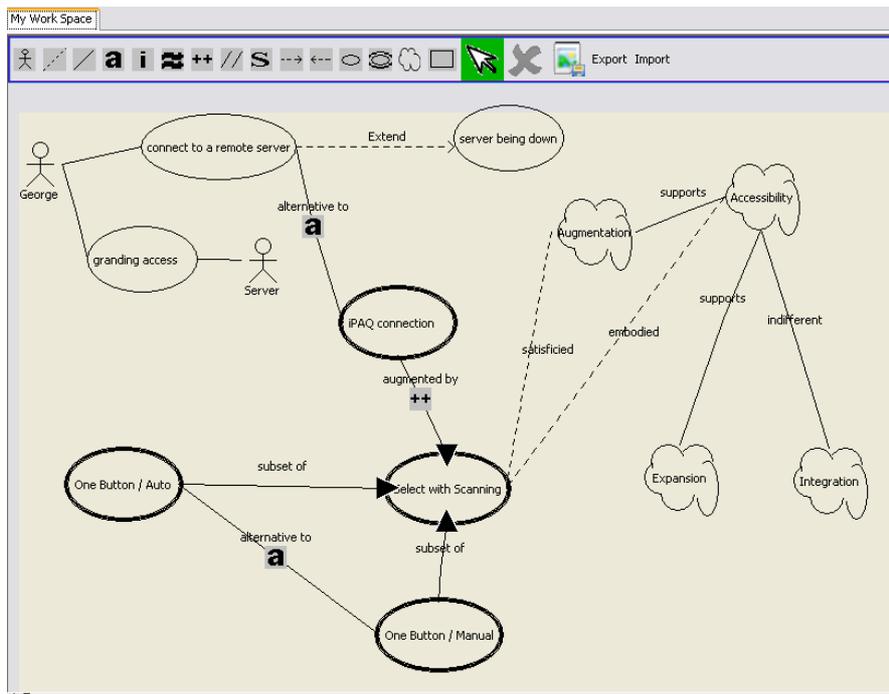


Figure 10. Designing an augmentation of a growth scenario



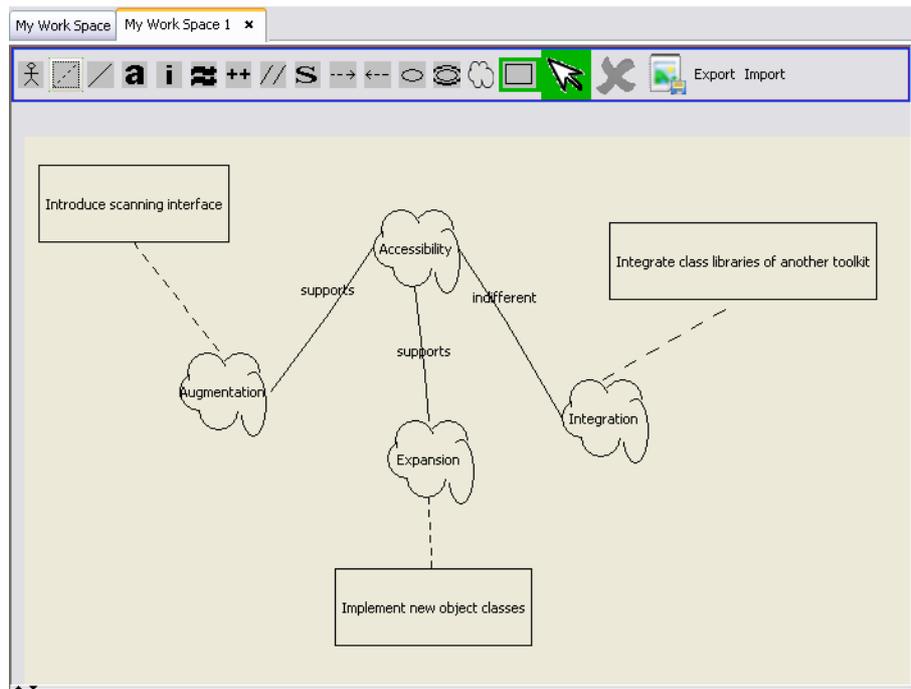


Figure 11. Quality models and the global execution context graph

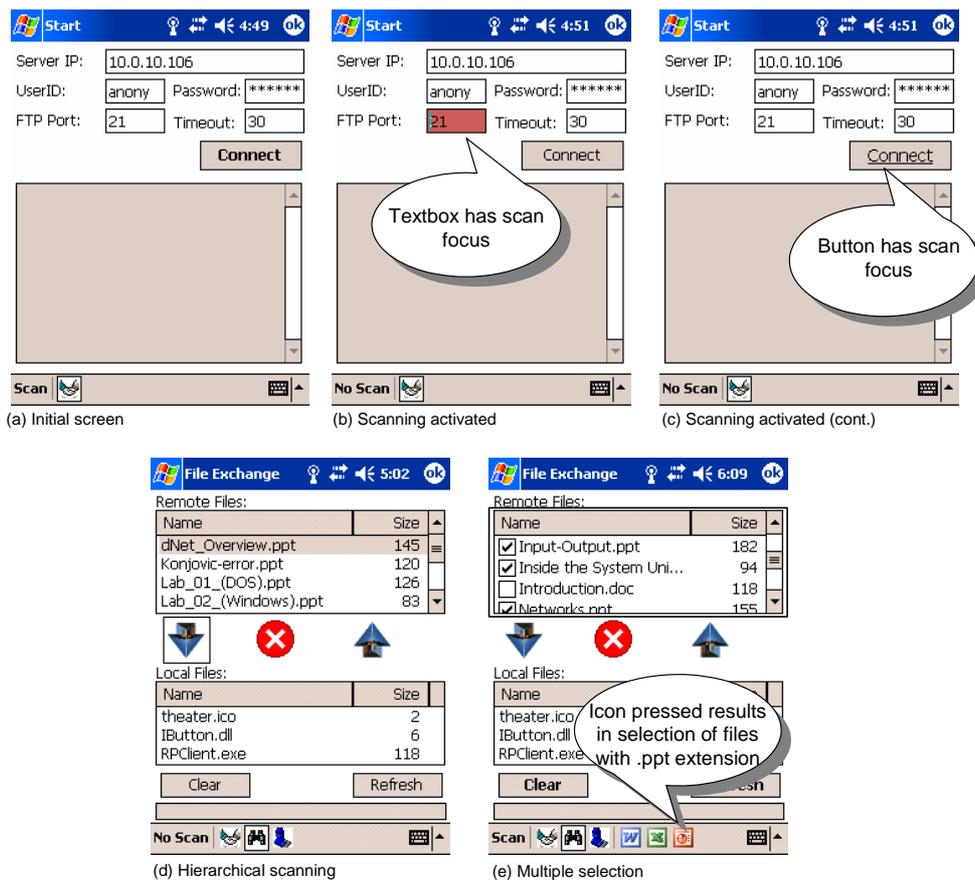


Figure 12. Examples of the scanning interface

5. Discussion

The work presented in this paper differs from recent related efforts both in terms of orientation and underlying perspective. In terms of orientation, our interest is to frame the problem of software execution across different non-functional contexts as an issue of software evolution. To this end, change in functional requirements is of course valid but it can only explain in part why modern information systems need to change. In fact, there is evidence to suggest that most of the changes in modern information systems do not concern functional components but their connections and interactions. This explains recent efforts aiming to frame change in the context of non-functional requirements and architectural quality attributes.

In terms of underlying perspective, the present work pursues a line of research, which is motivated by the fact that complexity of software is increasingly shifting from construction to evolution. In the past, the software design community addressed complexity in construction by devising abstractions (*i.e.*, components, visual notations, models and tools), which make construction-oriented artefacts first-class objects. In a similar vein, an approach to addressing complexity in software evolution could be focused on making the software evolution artefacts explicit through modelling them as first class objects. This is especially relevant for service-oriented architectures (SOA), aiming to appropriate the benefits of reusability and maintainability to foster the design of applications in an implementation independent manner using network services and connections between network services.

The NfRn provides insights towards this end by promoting a shift in the unit of analysis from task- or activity-level to task execution contexts. Then, designing software systems for execution across different non-functional contexts is conceived as specifying the system's global execution context. This requires an explicit account of platform-oriented non-functional requirements such as augmentation, expansion, integration and abstraction, which are considered as quality goals inscribed in a SOA. Moreover as software designers will increasingly be required to articulate the global execution context of a system's tasks, there is a compelling need for tools supporting the management of designated software evolution artefacts. In our work, this is facilitated by extending the use case notation widely employed for documenting functional requirements in a manner facilitating the construction and refinement of the tasks' global execution context graph.

The global execution context notation and the supporting tool have now been applied in a number of case studies and applications (see [44-46]) in addition to the initial validation in the Health Telematics domain [47], providing useful insight to managing change in interactive software. These experiences provide evidence to support

the claim that the basic vocabulary of the GeC and the method presented in this paper offer useful insight to modelling software design evolution necessitated either by new requirements or evolving requirements. The primary benefit of the method results from the fact that change becomes a first class design object modelled through designated growth scenarios that evolve from previously codified reference scenarios. Moreover, the GeCg as an artefact provides designers with useful information regarding:

- The range of alternative execution contexts considered appropriate at a point in time.
- The conditions which characterize activation/deactivation of growth scenarios; this entails an elaboration and justification of each of the relationships appearing in the graph.
- Guidance in the choice of what paths to traverse or walk through under specific conditions.
- Choice of suitable system architecture; for example relationships of the type alternative and augments designate the systems adaptable components, while the relationship type parallel points out adaptive features of the target implementation.

Consequently, the main contributions of the presented work are threefold. Firstly, we described a method for modelling change early in the development lifecycle. This is done by introducing a notation, which is simple and intuitive while resembling the vocabulary used by other popular notations such as UML. It is argued that using this notation to specify the current and anticipated contexts of use constitutes an improvement upon current practices. Specifically, the burden of using textual descriptions to codify goals (as in the case of RUP) is removed. Instead, visual constructs are used to codify design logic and rationale in a manner similar to other research proposals for visual goal-oriented requirements modelling [3-5]. Secondly, the method offers a frame of reference for considering scenarios as drivers for system evolution. This departs from contemporary views of scenario-based requirements engineering where scenarios are considered as static resources appearing at the beginning of a project and lasting until specifications or requirements are documented. In our work, scenarios remain 'live' and persistent resources driving future system evolution. Moreover, this is achieved in a systematic manner and it is documented using appropriate computer-based tools. Another contribution of the present work is that it is particularly suited to dealing with non functional requirements – such as adaptability, adaptivity, scalability and portability – which in contrast to functional requirements, are known to be hard to model and account for. This offers a perspective on scenario evolution, which is complementary to existing conceptions proposed in the relevant literature (e.g. [17]).

6. Summary and Future Work

In this paper, we have presented a method and a supporting tool for specifying the global execution context of computer-mediated tasks. Our motivation has been to make explicit the artefacts of evolution. Thus, our method considers evolution as a transformation from the current situation (codified through reference scenarios) to an envisioned situation (represented by semantically related growth scenarios). The links characterizing such transformations are a small set of scenario relationships such as alternate execution, concurrency, ordering, and set-oriented relationships between two scenarios, devised to encapsulate evolution as change of functional requirements as well as evolution as change in non-functional qualities. A system's global execution context can then be depicted as a visual construction, referred to as the global execution context graph, and can be populated by a supporting tool suite and transformed to XML.

Future work seeks to address several extensions both in the method and the i-GeC tool. In terms of methodological extensions, we are studying the development of a scenario specification language to formalize the description of scenarios. On the other hand several refinements of the tool suite are currently under development. Specifically, an on going activity seeks to expand the (currently primitive) user interface prototyping features supported by the tool so as to establish a link between scenarios (either reference or growth), their underlying rationale and their (possible) interactive embodiments. In this context, we are also exploring the possibility of linking the tool's outcome with existing task-based notations and model-based user interface engineering methods such as Teresa [32].

REFERENCES

- [1] D. R. Olsen, "Interacting in Chaos," *Interactions*, Vol. 6, No. 5, September-October 1999, pp. 42-54.
- [2] S. S. Anand, P. Kearney and M. Shapcott, "Generating Semantically Enriched User Profiles for Web Personalization," *ACM Transactions on Internet Technology*, Vol. 7, No. 4, October 2007.
- [3] E. Yu, "Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering," *Proceedings of the 3rd IEEE International Symposium on Requirements Engineering (RE'97)*, IEEE Computer Society, New York, 1997, pp. 226-235.
- [4] H. Solheim, F. Lillehagen, S. A. Petersen, H. Jorgensen and M. Anastasiou, "Model-Driven Visual Requirements Engineering," *The Proceedings of the 13th IEEE International Conference on Requirements Engineering*, IEEE Computer Society, New York, 2005, pp. 421-425.
- [5] K. Cooper, S. P. Abraham, R. S. Unnithan, L. Chung and S. Courtney, "Integrating Visual Goal Models into the Rational Unified Process," *Journal of Visual Languages and Computing*, Vol. 17, 2006, pp. 551-583.
- [6] M. E. Dashofy, A. Van der Hoek and N. R. Taylor, "A Comprehensive Approach for the Development of Modular Software Architecture Description Languages," *ACM Transactions on Software Engineering and Methodology*, Vol. 14, No. 2, 2005, pp. 199-245.
- [7] J. Mylopoulos, L. Chung and B. Nixon, "Representing and Using Non-Functional Requirements: A Process-Oriented Approach," *ACM Transactions on Software Engineering*, Vol. 18, No. 6, 1992, pp. 483-497.
- [8] L. Chung, B. Nixon, E. Yu and J. Mylopoulos, "Non-Functional Requirements in Software Engineering," Kluwer Academic Publishers, Boston, 1999.
- [9] P. Finger, "Component-Based Frameworks for E-Commerce," *Communications of the ACM*, Vol. 43, No. 10, 2000, pp. 61-66.
- [10] M. M. Lehman and J. F. Ramil, "Software Evolution and Software Evolution Processes," *Annals of Software Engineering*, Vol. 14, No. 1-4, 2002, pp. 275-309.
- [11] D. Thomas, "Agile Programming: Design to Accommodate Change," *IEEE Software*, Vol. 22, No.3, 2005, pp. 14-16.
- [12] K. Beck, "Embracing Change with Extreme Programming," *IEEE Computer*, Vol. 32, No. 10, 1999, pp. 70-77.
- [13] E. M. Shina and H. Gomaab, "Software Requirements and Architecture Modeling for Evolving Non-Secure Applications into Secure Applications," *Science of Computer Programming*, Vol. 66, No. 1, 2007, pp. 60-70.
- [14] L. Naslavsky, A. T. Alspaugh, J. D. Richardson and H. Ziv, "Using Scenarios to Support Traceability," *Proceedings of the 3rd International Workshop on Traceability in Emerging Forms of Software Engineering*, ACM Press, New York, 2005, pp. 25-30.
- [15] J. Cleland-Huang and K. C. Chang, "Event-Based Traceability for Managing Evolutionary Change," *IEEE Transactions on Software Engineering*, Vol. 29, No. 9, 2003, pp. 796-810.
- [16] V. Rajlich, "Modeling Software Evolution by Evolving Interoperation Graphs," *Annals of Software Engineering*, Vol. 9, No. 1-4, May 2000, pp. 235-248.
- [17] K. K. Breitman, J. C. S. P. Leite and M. D. Berry, "Supporting Scenario Evolution," *Requirements Engineering*, Vol. 10, No. 2, May 2005, pp. 112-131.
- [18] D. B. Petriu, D. Amyot, M. Woodside and B. Jiang, "Traceability and Evaluation in Scenario Analysis by Use Case Maps," In: S. Leue and T. J. Systa, Ed., *Scenarios: Models, Transformations and Tools (Lecture Notes in Computer Science)*, Springer-Verlag, Berlin/Heidelberg, Vol. 3466, 2005, pp. 134-151.
- [19] J. Hammer and M. Schneider, "The GenAlg Project: Developing a New Integrating Data Model, Language, and Tool for Managing and Querying Genomic Information," *SIGMOD Record*, Vol. 33, No. 2, 2004, pp. 45-50.
- [20] I. Jacobson, M. Christerson, P. Jonsson and G. Overgaard, "Object-Oriented Software Engineering – A Use Case Driven Approach," Addison-Wesley, White Plains, 1992.
- [21] G. Avellis, "CASE Support for Software Evolution: A

- Dependency Approach to Control the Change Process,” *Proceedings of the 5th International Workshop on Computer-Aided Software Engineering*, IEEE Computer Society, New York, 1992, pp. 62-73.
- [22] T. Mens and T. D’Hondt, “Automating Support for Software Evolution in UML,” *Automated Software Engineering*, Vol. 7, No. 1, 2000, pp. 39-59.
- [23] T. Kosar, E. P. M. Lopez, A. P. Barrientos and M. Mernik, “A Preliminary Study on Various Implementation Approaches of Domain-Specific Language,” *Information and Software Technology*, Vol. 50, No. 5, April 2008, pp. 390-405.
- [24] J. Greenfield and K. Short, “Software Factories – Assembling Applications with Patterns, Frameworks, Models & Tools,” John Wiley & Sons, New York, 2004.
- [25] B. Myers, “User Interfaces Software Tools,” *ACM Transactions on Human-Computer Interaction*, Vol. 12, No. 1, 1995, pp. 64-103.
- [26] J. Heer, S. Card and J. Landay, “Prefuse: A Toolkit for Interactive Information Visualization,” *Proceedings of ACM CHI*, ACM Press, New York, 2005, pp. 421-430.
- [27] B. B. Bederson, J. Grosjean and J. Meyer, “Toolkit Design for Interactive Structured Graphics,” *IEEE Transactions on Software Engineering*, Vol. 30, No. 8, 2004, pp. 535-546.
- [28] E. Adar, “GUESS: A Language and Interface for Graph Exploration,” *Proceedings of the ACM Conference on Human Factors in Computing Systems*, ACM Press, New York, 2006, pp. 791-800.
- [29] M. Barbacci, R. Ellison, A. Lattanze, J. Stafford, C. Weinstock and W. Wood, “Quality Attribute Workshops,” 2nd Edition, Carnegie Mellon Software Engineering Institute, Pittsburgh, 2002. <http://www.sei.cmu.edu/pub/documents/02.reports/pdf/02tr019.pdf>
- [30] L. Bass, P. Clements and R. Kasman, “Software Architecture in Practice,” Addison-Wesley, White Plains, 1998.
- [31] L. Chung and N. Subramanian, “Adaptable Architecture Generation for Embedded Systems,” *The Journal of Systems and Software*, Vol. 71, No. 3, 2004, pp. 271-295.
- [32] G. Mori, F. Paternò and C. Santoro, “Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions,” *IEEE Transactions on Software Engineering*, Vol. 30, No. 8, 2004, pp. 507-520.
- [33] M. Salehie and L. Tahvildari, “Self-Adaptive Software: Landscape and Research Challenges,” *ACM Transactions on Autonomous and Adaptive Systems*, Vol. 4, No. 2, 2009.
- [34] M. Barbacci, M. Klein, T. Longstaff, C. Weinstocket, “Quality Attributes,” Software Engineering Institute, Carnegie Mellon University, Pittsburgh, 1995. <http://www.sei.cmu.edu/publications/documents/95.reports/95.tr.021.html>
- [35] E. Folmer and J. Bosch, “Architecting for Usability: A Survey,” *Journal of Systems and Software*, Vol. 70, No. 1-2, 2004, pp. 61-78.
- [36] A. MacLean, V. Bellotti and S. Shum, “Developing the Design Space with Design Space Analysis,” In: P. F. Byerley, P. J. Barnard and J. May, Ed., *Computers, Communication and Usability: Design Issues, Research and Methods for Integrated Services*, Elsevier, Amsterdam, pp. 197-219, 1993.
- [37] J. Lee and K.-Y. Lai, “What’s in Design Rationale?” In: T. P. Moran and J. M. Carroll, Ed., *Design Rationale: Concepts, Techniques and Use*, Lawrence Erlbaum Associates, Mahwah, 1996.
- [38] J. S. Olson and T. P. Moran, “Mapping the Method Muddle: Guidance in Using Methods for User Interface Design,” In: M. Rudisill, C. Lewis, P. B. Polson and T. D. McKay, Ed., *Human-Computer Interface Design: Success Stories, Emerging Methods, and Real-World Context*, Morgan Kaufmann Publishers, San Francisco, 1996, pp. 101-121.
- [39] D. Akoumianakis, A. Savidis and C. Stephanidis, “Encapsulating Intelligent Interactive Behavior in Unified User Interface Artifacts,” *Interacting with Computers*, Vol. 12, No. 4, 2000, pp. 383-408.
- [40] J. C. S. P. Leite, J. H. Doorn, G. D. S. Hadad and G. N. Kaplan, “Scenario Inspections,” *Requirements Engineering*, Vol. 10, 2005, pp. 1-21.
- [41] K. Weidenhaupt, K. Pohl, M. Jarke and P. Haumer, “Scenarios in System Development: Current Practice,” *IEEE Software*, Vol. 15, No. 2, 1998, pp. 34-45.
- [42] C. Potts, K. Takahashi and A. Anton, “Inquiry-Based Requirements Analysis,” *IEEE Software*, Vol. 11, No. 2, 1994, pp. 21-32.
- [43] C. Rolland and C. B. Achour, “Guiding the Construction of Textual Use Case Specifications,” *Data & Knowledge Engineering*, Vol. 25, No. 1-2, 1998, pp. 125-160.
- [44] D. Akoumianakis and I. Pachoulakis, “Scenario networks: Specifying User Interfaces with Extended Use Cases,” In: P. Bozaris and E. N. Houstis, Ed., *Advances in Informatics (Lecture Notes in Computer Science)*, Springer-Verlag, Berlin/Heidelberg, Vol. 3746, 2006, pp. 491-501.
- [45] D. Akoumianakis, A. Katsis and N. Bidakis, “Non-functional User Interface Requirements Notation (NfRn) for Modeling the Global Execution Context of Tasks,” In: K. Coninx, K. Luyten and K. A. Schneider, Ed., *Task Models and Diagrams for Users Interface Design (Lecture Notes in Computer Science)*, Springer-Verlag, Berlin/Heidelberg, Vol. 4385, 2006, pp. 259-274.
- [46] D. Akoumianakis, G. Vellis, D. Kotsalis, G. Milolidakis and N. Vidakis, “Experience-Based Social and collaborative Performance in an ‘Electronic Village’ of Local Interest: The eKoNEΣ Framework,” In: J. Cardoso, J. Cordeiro and J. Filipe, Ed., *ICEIS’2007 – 9th International Conference on Enterprise Information Systems, Volume HCI*, INSTICC, Funchal, 2007, pp. 117-122,
- [47] D. Akoumianakis and C. Stephanidis, “Blending Scenarios and Informal Argumentation to Facilitate Universal Access: Experience with the Universal Access Assessment Workshop Method,” *Behaviour & Information Technology*, Vol. 22, No. 4, 2003, pp. 227-244.