AGILE

IST-2001-32747
Architectures for Mobility
www.pst.ifi.lmu.de/projekte/agile

AGILE Brochure
Building Architectures for Mobility

Version: 1
Date: 2005/04/29

Project funded by the European Community under the "Information Society Technologies" Programme (1998—2002)

Contents

1 Introduction

Architecture-based approaches have been promoted as a means of controlling the complexity of system construction and evolution, namely for providing systems with the agility required to operate in turbulent environments and to adapt very quickly to changes in the enterprise world. Recent technological advances in communication and distribution have made mobility an additional factor of complexity, one for which current architectural concepts and techniques are not prepared for. The EU project Architectures for Mobility (AGILE) addresses this new level of complexity by developing an architectural approach in which mobility aspects can be modelled explicitly and mapped on the distribution and communication topology made available at physical levels. The whole approach is developed over a uniform mathematical framework based on process algebra and graph-oriented techniques that support sound methodological principles, formal analysis, and refinement across levels of development. Application areas of AGILE include E-business, telecommunications, wireless applications, traffic control systems, and decision support systems which need to collect global information.

The research in AGILE is based on five views. The first view is architectures for mobility, where notations and abstractions for describing mobile systems are developed. In the second view, these abstractions are made available to the software engineer through UML extensions and support for development processes. The notations and abstractions are based on a firm mathematical semantics which gives rise to validation and verification techniques. Finally, tools are developed supporting the software engineer in his task of modelling, programming, and analysing mobile systems.

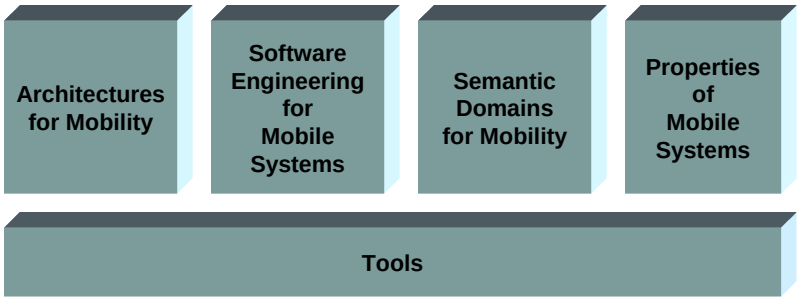

Figure 1: The five views on AGILE.

2 Architectures for Mobility

AGILE has a very general view on mobile systems which includes mobile computing as well as mobile computation. Mobile systems in AGILE are systems that are at some locations and include systems representing computations, i.e. programs, objects, and physical objects, like mobile phones or people. Locations can be networks, computers, mobile objects, as well as physical locations, like airports and planes as in Fig. 2. This view on mobility includes highly dynamic distributed systems where the distribution and interconnection of systems change (e.g., by moving the system to another place); it also includes systems where the software running on these systems change over time (e.g., resource intensive software might travel to a system with sufficient resources). For a mobile system to be at a location has several implications. One implication is that, depending on the location, the system reacts differently. This is called context awareness of mobile systems. There could be several reasons for such a context sensitive behaviour: For example, systems may have access to different set of resources and services depending on the location of the system.

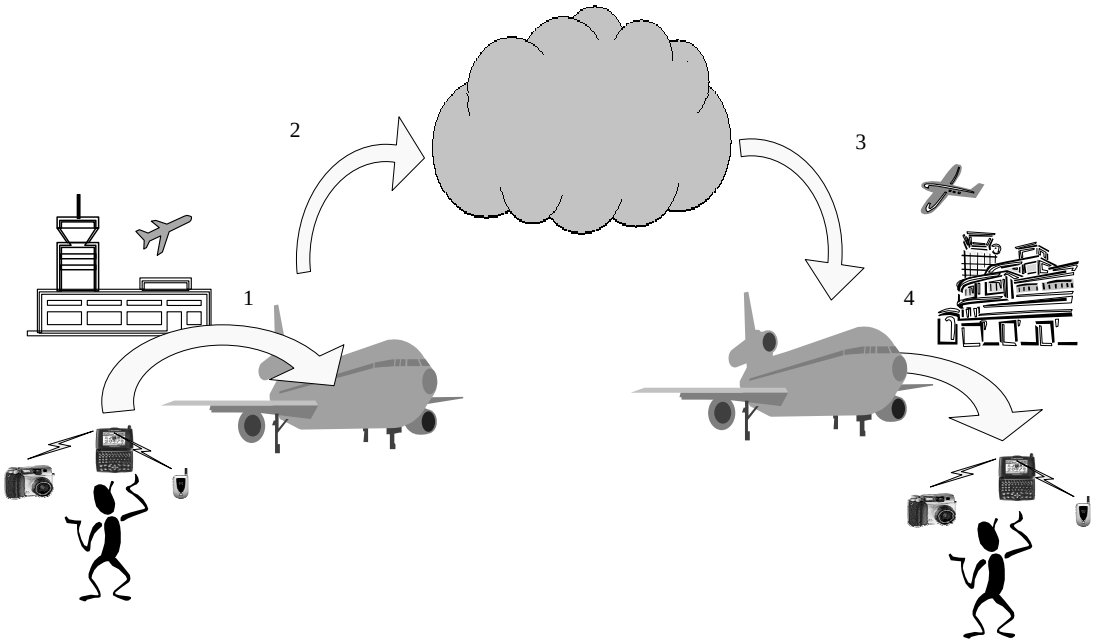

Figure 2: An example of nested mobile networks.

On the architectural level, mobility is treated in AGILE as an own dimension similar to computation and coordination. Computational units, i.e. components, contain state and activities that can be distributed over a network of locations. The components are connected using connectors. Some connectors coordinate the way components interact. For instance, in a financial system, a customer may interact with an account via a standard package that only provides for deposits and withdrawals. Upgrading the customer to a more flexible account with overdraft and saving facilities should be a matter of just replacing the connector without having to re-program the components. Similarly, other connectors are responsible for managing the mobility aspect of components, that is, where the constituents of components reside — i.e., where their state is located and their activities are performed — and how this may change. For instance, different rules apply to the way a customer can interact with a bank account, depending on the location, i.e. if the customer is at a branch of the bank, at an ATM, or using the Web. Each of these modes should be captured by a different connector. An extension of the syntax and the semantics of the program design language CommUnity was developed in order to support the design of components that operate and interact in a network of locations, as well as the connectors that capture the coordination and distribution/mobility aspects of their behaviour.

Complementary to CommUnity, CASL architectural specifications allow one to describe how a system is built from units in contrast to how the system is structured. Thus a CASL architectural specification of a system describes which parts of the system can be developed independently and how these independently developed parts can be combined to form the complete system.

3 Software Engineering for Mobile Systems

The results of the work on CommUnity provided a formal framework for the design of the Component Development Environment (CDE). The CDE is a framework of structuring Java applications to allow the separation of concerns as found in CommUnity. In the CDE, components in Java (i.e. Java objects) represent the computational units which are connected by coordination and location laws. These laws determine the behaviour of the components based on the coordination patterns they are involved in and their locations. For example, components may behave differently depending on where they are located. While the computational

units are quite stable, the coordination and location laws change frequently. This architectural style has been used extensively in projects of the Portuguese company ATX Software SA.

An extension of UML, called Mobile UML, has been defined to model systems with mobility. Mobile UML comprises extensions of class, activity, and sequence diagrams, and state machines. Classes can have stereotype «mobile», «location» or «mobile location». Mobile objects are at locations and mobile locations are locations which are themselves mobile objects. Examples are a passenger who is located at a plane which again is located at an airport (cf. Fig. 2). Location based activity diagrams are used to visualise how activities change the location of objects, e.g., Fig 3 shows this notation for the boarding activity of a passenger. UML state machines have been extended with specific actions for the description of mobility and network management, e.g., the creation of network nodes and object migration.

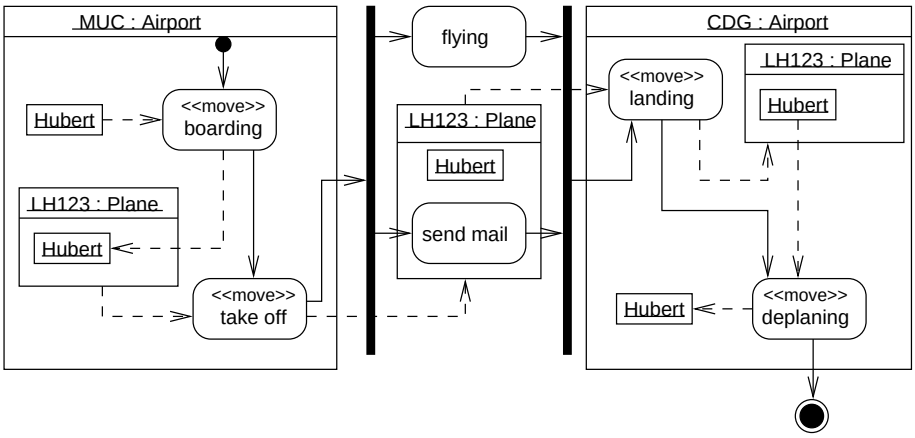

Figure 3: The airport example using the location cantered activity diagram notation of Mobile UML.

Furthermore, three pragmatic approaches have been developed in AGILE extending standard software development processes to cope with mobile systems development. Software development for mobile systems nowadays takes place in a highly agile environment. It is important that the software is produced quickly and that the resulting software is reliable. What complicates matters is, that when software development starts, the requirements are not fully understood and that during the lifetime of the software project, requirements are likely to change from the feedback resulting from early experiments with the software.

In the first approach, formal methods are combined with the test-driven development practice of Extreme Programming. The property-driven development approach applies the techniques of the first approach using executable UML models instead of program code. Finally, the third approach investigates the possibility of using code generation techniques to build executable UML models with CASE tools.

4 Semantic Domains for Mobile Systems

Within AGILE several approaches to the semantics of mobile systems have been developed. These approaches differ in the aspects of mobile systems they focus on, in the properties that are expressible over the semantic models, and what kind of analysis techniques can be applied to them, e.g., techniques from process algebra, graph transformation, tile logic, or universal algebra.

KLAIM (Kernel Language for Agent Interaction and Mobility) is an experimental programming language specifically designed to model and to program distributed concurrent applications with code mobility. The language relies on the concept of tuple space to store data. Tuples can be read and updated. Tuple spaces are placed on nodes that are part of a net. Each node contains a single tuple space and processes in execution; a node can be accessed through its address. KLAIM processes may run concurrently, both at the same

node or at different nodes. KLAIM has been extended in several ways to cope with specific problems of mobile systems. One extension, OpenKlaim, was specifically designed for enabling users to give more realistic accounts of open systems. Open systems are dynamically evolving structures: new nodes can get connected or existing nodes can disconnect. Connections and disconnections can be temporary and unexpected. Another extension, STOcKLAIM, makes it possible to integrate the modelling of quantitative aspects of mobile systems, e.g., performance, with the functional specification of such systems by associating the average time to an action of a process in a KLAIM network.

Synchronised Hyperedge Replacement (SHR) systems are an extension of graph transformations with a simple synchronisation mechanism inspired by the tile model. They are used as a semantic base to encode object systems and transformations between them. SHR systems are very well suited for the semantics of diagram based systems, like the UML. Both, KLAIM and SHR have been used to give an operational semantics for CommUnity.

To apply the architectural specifications of CASL to mobile systems, an institution for specifying distributed systems has been developed. Signatures contain sorted input and output variables and actions. Specifications are similar to CommUnity designs. Models are labelled transition systems where the states are composed of the valuations of the input and output variables to values over some data algebra. Transitions are relations between states and labelled by actions. For mobility, the data algebras contain a special sort *location*, and mobility means to change the interpretation of variables of sort *location*.

5 Expressing Properties of Mobile Systems

To express properties of mobile systems, several logics have been developed on different levels of abstraction (cf. Fig. 4). Based on Lamport's Temporal Logic of Actions (TLA), Mobile TLA (MTLA) extends TLA

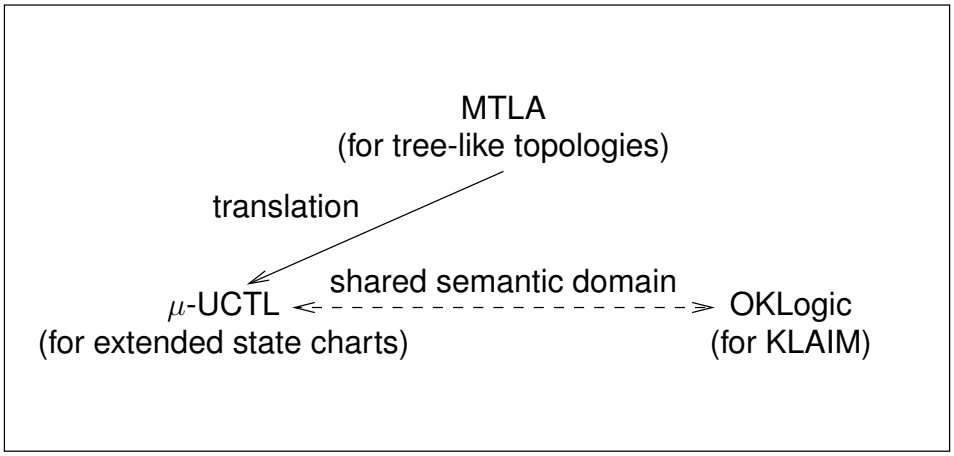

Figure 4: Relationship between logics for mobility developed AGILE.

with spatial modalities. With MTLA it becomes possible to express the mobility of objects with respect to their containing locations. For example, one can express that an object resides at a certain location or will be residing at that location in the future.

μ-UCTL is a state/event-based temporal logic which is oriented towards a natural description of dynamic properties of UML models. This logic allows one to specify the basic properties that a runtime system configuration should satisfy, and combines these basic predicates with logic and temporal operators to take into consideration also the events performed by the system when evolving from one system configuration to another. Doubly labelled transition systems are the semantic domain for μ-UCTL. The logic is supported by a prototypical verification environment developed in the AGILE project around the "on the fly" UMC

model checker. MTLA formulas can be translated into μ-UCTL formulas which then can be checked with the UMC model checker. Based on the semantic domain of μ-UCTL, OKLogic is a logic for expressing properties of OpenKlaim nets.

6 Tools

The tools being developed in AGILE support the software engineer by helping him/her to model mobile systems, to write better software, and to analyse properties of software.

Modelling tools The CommUnity Workbench allows to experiment with mobile system architectures written in CommUnity, while ArgoMobile facilitates the modelling of mobile systems using the UML extensions for mobility developed in AGILE (cf. Fig. 5). It includes an extension of class diagrams to model mobility and location laws and extensions of activity diagrams. From class diagrams with location laws it is possible to generate input for the Component Development Environment. ArgoMobile is based on the open-source UML modelling tool ArgoUML[1].

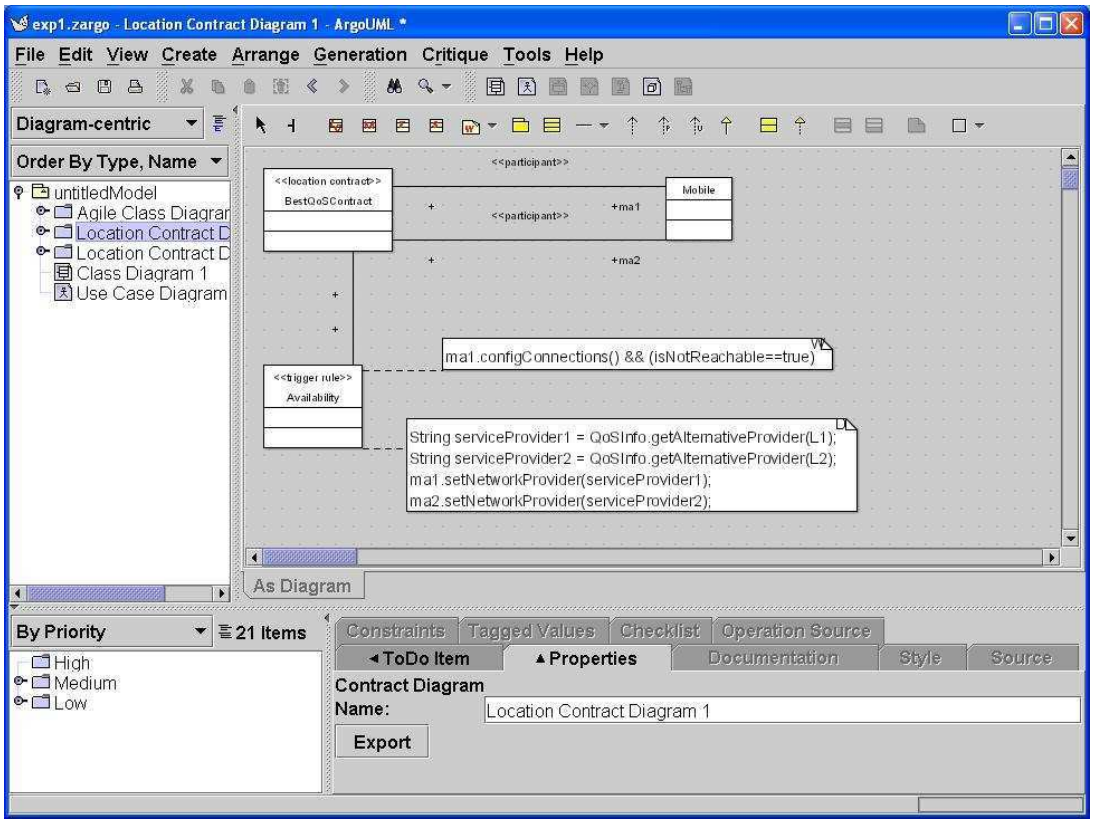

Figure 5: Screenshot of the ArgoMobile Tool.

Programming tools The Component Development Environment allows one to develop software by employing the architectural concepts of AGILE using coordination- and location laws. X-KLAIM is a programming language that extends KLAIM with high-level constructs and strong mobility. A compiler translates X-KLAIM programs into Java programs that rely on the Klava package (KLAIM in JAVA) which implements all the functionalities for a run-time systems for KLAIM.

[1] www.argouml.org

Analysis tools For model-check mobile systems, HAL, for the formal validation of logical and behavioural properties of for π-calculus agent, and UMC, an "on the fly" μ-ACTL+ model checker for UML state charts, have been implemented. KlaiML is a framework that permits analysing KLAIM systems and verifying the logical properties of mobile and distributed systems.

7 Partners

AGILE is funded by the European Commission as part of the Global Computing Initiative[2] and is part of the cluster *Language and Programming Environments* together with Mikado (Mobile Calculi based on Domains)[3], Degas (Design Environments for Global Applications)[4], and Pepito (Peer-to-Peer: Implementation and Theory)[5]. Partners in AGILE are

- Ludwig-Maximilians-Universität München, Institut für Informatik, Munich, Germany (Co-ordinator), headed by Martin Wirsing
- Università di Pisa, Dipartimento di Informatica, Pisa, Italy, headed by Andrea Corradini
- Università di Firenze, Dipartimento di Sistemi ed Informatica, Florence, Italy, headed by Rocco de Nicola
- Istituto di Scienza e Tecnologie della Informazione "A. Faedo", Pisa, Italy, headed by Stefania Gnesi
- ATX Software SA, Lisbon, Portugal, headed by Luís Andrade
- Fundação da Faculdade de Ciências da Universidade de Lisboa, Lisbon, Portugal, headed by Antónia Lopes
- Warsaw University, Institute of Informatics, Warsaw, Poland, headed by Andrzej Tarlecki
- and University of Leicester, Department of Computer Science, Leicester, UK, headed by José Luiz Fiadeiro.

8 Further Information

Further information can be obtained from the AGILE Web-site[6], which also contains references to publications and tools, or by sending e-mail to `Martin.Wirsing@ifi.lmu.de`.

[2] `http://www.cordis.lu/ist/fetgc.htm`
[3] `http://mikado.di.fc.ul.pt/`
[4] `http://www.omnys.it/degas`
[5] `http://www.sics.se/pepito/`
[6] `http://www.pst.ifi.lmu.de/projekte/agile`