SPECIFICATION AND VERIFICATION
OF MOBILE SYSTEMS

Simone Semprini

Ph.D. Thesis
Università di Pisa
Facoltà di Scienze M.F.N.
Dipartimento di Informatica
Abstract. Classical and established habits in the use of computers are being complemented, and it is not difficult to foresee a partial replacement, by new trends, both technological and social, and by new ways of conceiving computation and applications. Pervasiveness and flexibility are at the bases of the development of new kind of devices like Personal Digital Assistants (PDAs), Internet enabled cellular phones, and the outcomes that could flourish from the appealing and futuristic convergence of these two families of appliances. Besides, Internet working is definitely stressing the meaning of the term distributed application especially with regard to the operating environment that applications must be able to cope with. Multiple administrative domains, each one with its own security and resource access policies, the most varied execution platforms, heterogeneity of communications models, a strong interest in quality assurance measures, mixed networks that integrate a fixed structure with wireless ad hoc networks are some of the elements that drive the design and the implementation of a new class of applications.

Such a shift in the framework of reference for the development of network and stand-alone applications poses great challenges to designers and implementers, as witnessed by the blossoming of research works from both university and industry. This thesis has its collocation and rationale in the research trend that considers mobility an answer to the mentioned challenges. We look at the experiences conducted so far by many researchers and develop our own solution to the problem of specifying and verifying mobile distributed systems. A formal approach and a strong commitment to architectural aspects of the systems of interest animate our proposal.

The adoption of a formal approach allows us to define unambiguously the characteristic of our systems and poses the basis for the rigorous verification of the properties they should exhibit. The properties we want to express and reason about are those that define the coordination patterns ruling the interactions of sets of entities. We take the move from a multimodal logic tailored for the description of the evolution of distributed systems. Spatial modalities are the constituents of formulae that express properties of distributed states in terms of the ones exhibited by single components; temporal modalities provides means of stating liveness and safety conditions for systems evolution. Communications in our logic are asynchronous to reflect a decoupled style of interaction among the components of a system; this style, indeed, better fits with the poor assumptions that can be made on net infrastructure in our reference setting. Among the appealing features of our proposal, there are the compositional nature of specification and a refinement based approach that let designers cope with the specification task gradually and incrementally. The commitment to a declarative formalism, instead of a process algebraic or programmative one, has its rationale in our belief that it makes more intuitive to express and easier to read the properties of interest.

To take into the proper consideration aspects like location awareness and transparency, multiple security policies, separation of computation and coordination, and flexibility in communications and mobility we defined a model that sketches the general structure of a mobile distributed system. Systems are composed of mobile agents that roam in a net of localities and communicate each other via asynchronous message passing. Each locality has one stationary controlling entity, called guardian, that conceptually
centralizes the activities related to the enactment of the security policy of the locality with regard to the actions of mobile agents that interact with it. The coordination of the guardians’ activities defines a dynamic global security policy in terms of the security policies of each location. Guardians also manage the routing of messages and mobile agents dealing with possible failures due to security violations or infrastructure malfunctioning. Operating on the routing decisions of the guardians, we can integrate in our systems any topological structure of the network without committing to a particular one. Guardians also provide an identification mechanism that allows specifying an entity through a profile, i.e. a set of properties. This mechanism seamlessly accommodates many particular choices among which physical and logical names, unicast, multicast or broadcast communications, and service discovery. Moreover, it can be exploited also to cope with Quality of Service (QoS) concerns, for example in the description of migrating applications that base the choice of a target locality on some minimal guarantees on the values of a set of qualitative parameters like throughput and latency.

Using our logic we formalize the model and prove the basic structural properties of communications and mobility. The result of this formalization is the starting point for the specification of any system that proceeds from the reference model to the final design through a chain of refinement steps. Each step adds some details to a particular aspect of the system under analysis and all the refinements can be composed back to obtain the overall description of the system. We depict a proper structure of the specifications that allows clearly identifying and separating computation and coordination; moreover, the compositional nature of our approach helps to factorize specifications, thus fostering their reuse, and eases the intervention of different expertises in the design of a system.

Since the introduction of formal methods in the design is too often seen as a burden on the designers, and with the intent of easing a successful application of our methodology, we enrich our proposal with the adjunct of a tool that provides designers with some support in proving the properties of their systems. We based this tool on a theorem prover, instead of a model checker, because of the need to cope with infinite state specifications and the possibility to produce proof traces that third parties can check. Moreover, this choice allowed us to support designers with a tool directly based on our logic, which we claim to be intuitive and close to designers’ domain knowledge even because of the possibility of refining specifications to define proper predicates and terms to directly map domain elements into the specifications. The tool comes equipped with a set of tactics and pre-proved theorems that can be enriched at any proof session to form a library that grows depending on the particular needs and domains of interest of the designer.

An initial exploration on a possible implementation of our model concludes this thesis. The aim of an implementation of the model would be that of making easier the passage from design to code, indeed, all the concepts used to build the formal description of the system would find an implementative counterpart.
Index

AKNOWLEDGMENTS ........................................................................................................... 7
INDEX OF FIGURES ........................................................................................................... 8
INDEX OF TABLES ............................................................................................................. 9
1 INTRODUCTION ............................................................................................................ 10
2 MOBILITY ....................................................................................................................... 12
  2.1 APPLICABILITY SCENARIOS AND MAIN ISSUES ................................................. 12
  2.2 MODELS FOR MOBILITY: A SURVEY ................................................................. 15
    2.2.1 Mobile UNITY .................................................................................................... 17
      2.2.1.1 Mobile UNITY applications ........................................................................ 19
      2.2.1.1 Formalization of Mobile IP .......................................................... 20
      2.2.1.2 Code mobility ......................................................................................... 21
      2.2.1.3 Middleware for mobility ............................................................................ 22
    2.2.2 Mobile Ambients ................................................................................................. 24
      2.2.2.1 Mobile Ambients dialects ......................................................................... 26
      2.2.2.1.1 Safe Ambients ..................................................................................... 26
      2.2.2.1.2 Boxed Ambients .................................................................................. 28
    2.2.3 KLAIM ............................................................................................................... 30
      2.2.3.1 KLAIM’s dialects ....................................................................................... 32
      2.2.3.1.1 Open KLAIM ...................................................................................... 32
      2.2.3.1.2 Hot KLAIM ......................................................................................... 33
      2.2.3.1.3 O’KLAIM ............................................................................................ 34
      2.2.3.1.4 XKLAIM ............................................................................................. 35
3 FORMAL METHODS IN SOFTWARE ENGINEERING .................................................... 37
  3.1 EXTENT AND SCOPE OF FORMAL METHODS APPLICATION ......................... 37
  3.2 TOOLS TO SUPPORT THE REASONING ON SYSTEMS ...................................... 39
    3.2.1 Model Checkers ............................................................................................... 39
    3.2.2 Interactive Theorem Provers ............................................................................ 40
    3.2.3 Automated Theorem Provers ........................................................................... 40
4 MOBAdTL: A MODEL AND A METHODOLOGY FOR MOBILITY .............................. 42
  4.1 SOME METAPHORS TO TALK ABOUT MOBILITY ............................................... 42
  4.2 THE LOGIC .............................................................................................................. 45
    4.2.1 Distributed States Logic: DSL(x) ................................................................. 45
    4.2.2 Distributed States Temporal Logic with events: DSTL(x) ....................... 50
    4.2.3 A comparison with related logics ................................................................. 56
  4.3 REFINEMENT AS A METHODOLOGICAL CHOICE .............................................. 59
  4.4 THE AXIOMATISATION ......................................................................................... 61
    4.4.1 The requirements of the model ..................................................................... 61
    4.4.2 The reference model ..................................................................................... 65
  4.5 A COMPARISON WITH RELATED MODELS ......................................................... 70
    4.5.1 Architectural choices .................................................................................... 70
    4.5.2 Mobility ......................................................................................................... 71
    4.5.3 Communications ............................................................................................ 72
    4.5.4 Security ......................................................................................................... 73
5 VERIFICATION OF MOBAdTL SYSTEMS ................................................................. 76
  5.1 THE THEOREM PROVER ISABELLE ....................................................................... 76
    5.1.1 Defining a logic .............................................................................................. 76
      5.1.1.1 Rule declarations .................................................................................... 77
      5.1.1.2 Type and constant declarations ......................................................... 79
    5.1.2 Performing proofs ......................................................................................... 81
  5.2 MARK: MOBAdTL REASONING KIT ................................................................. 85
    5.2.1 Coding the logics ......................................................................................... 85
      5.2.1.1 DSL: coding DSL(x) .............................................................................. 86
      5.2.1.2 DSL tactics and proofs ........................................................................... 87
5.2.1.3 dDSTL: coding ADSTL(x) ................................................................. 94
5.2.1.4 dDSTL tactics and proofs ................................................................. 95
5.2.2 Coding the Mob_adl theories .............................................................. 101
5.2.3 Mob_adl proofs ...................................................................................... 105
5.2.3.1 Handling quantifiers and acquiring assumptions ................................. 105
5.2.3.2 Tactics and proofs .............................................................................. 108
6 EXAMPLES ................................................................................................. 115
6.1 WORKING WITH PROFILE RESOLUTION .................................................. 115
6.1.1 Unicast and multicast communications .................................................. 115
6.1.2 Explicit naming ...................................................................................... 118
6.2 AN ABSTRACT INTERFACE TO SERVICE DISCOVERY ............................ 121
6.3 DIFFERENT KINDS OF MOBILITY .............................................................. 128
6.3.1 Mobile devices ....................................................................................... 128
6.3.2 Migrating agents ................................................................................... 130
7 EXPLORATIONS ON THE IMPLEMENTATION OF THE MODEL .................... 135
8 CONCLUSIONS ............................................................................................. 138
BIBLIOGRAPHY ............................................................................................... 140
APPENDIX A ..................................................................................................... 147
THE LOGIC THEORIES ..................................................................................... 147
PC.thy............................................. 147
DSL.thy........................................ 148
DSTL.thy...................................... 150
THE MODEL THEORIES .................................................................................. 153
mob_adl_support.thy............................................ 153
mob_adl_A0.thy........................................... 155
mob_adl_C0.thy........................................... 156
mob_adl_G0.thy........................................... 157
mob_adl_A1.thy........................................... 157
mob_adl_C1.thy........................................... 158
mob_adl_G1.thy........................................... 160
mob_adl_refModel.thy................................. 161
THE THEORIES OF THE EXAMPLES ............................................................. 162
mob_adl_multicast_support.thy.................. 162
mob_adl_multicast_G.thy......................... 162
mob_adl_multicast_refModel.thy............. 162
mob_adl_unicast_support.thy.................. 163
mob_adl_unicast_C.thy......................... 163
mob_adl_unicast_G.thy......................... 163
mob_adl_unicast_G.thy......................... 163
mob_adl_unicast_refModel.thy............. 164
mob_adl_prof_name_support.thy............ 165
mob_adl_prof_name_G.thy.................... 165
mob_adl_prof_name_refModel.thy.......... 166
mob_adl_services_support.thy............ 166
mob_adl_services_A1.thy.................... 167
mob_adl_services_C1.thy.................... 168
mob_adl_services_G1.thy.................... 168
mob_adl_services_C2.thy.................... 169
mob_adl_services_G2.thy.................... 169
mob_adl_services_refModel.thy........ 170
mob_adl_mobDevices_support.thy........ 171
mob_adl_mobDevices_G.thy................ 171
mob_adl_mobDevices_refModel.thy..... 171
mob_adl_migAgents_support.thy........ 172
mob_adl_migAgents_A.thy................ 172
mob_adl_migAgents_C.thy................ 172
mob_adl_migAgents_G.thy................ 173
APPENDIX B .......................................................................................................................... 174

THE LOGIC THEORIES .............................................................................................................. 174
  PC.ML ........................................................................................................................................... 174
  DSL_support.ML .......................................................................................................................... 178
  DSL.ML ........................................................................................................................................ 178
  DSTL.ML ..................................................................................................................................... 186

THE MODEL THEORIES ............................................................................................................. 191
  mob_adtl_support.ML .................................................................................................................. 191
  mob_adtl_refModel.ML ............................................................................................................... 192

THE THEORIES OF THE EXAMPLES ........................................................................................ 205
  mob_adtl_prof_name_refModel.ML ............................................................................................ 205
  mob_adtl_services_refModel.ML ............................................................................................... 210
  mob_adtl_migAgents.ML ............................................................................................................ 212
Acknowledgments

Many elements concur to define the context in which a work like this thesis is conceived and realized, and many persons play a role in this context each one bringing his/her own contribution in terms of both professional and personal support. I do feel I owe a lot to these persons and here I have the pleasure to thank them all.

I would like to thank my advisor Prof. Carlo Montangero because he is the one who made me meet software engineering, and for having given me an inspiring and friendly professional setting. With regard to this, a particular mention goes to Prof. sa Laura Semini whose friendly and graceful help meant so much in my struggle with modal logics. Many other people in my department had some influence on my work and gave me some support; among them, I would like to thank Prof. Gianluigi Ferrari.

My parents deserve a particular thank for all the efforts and sacrifice they made and because they believed in me and never failed to support me.

Having someone besides you when you are trying to understand what to do with your life is invaluable. I had this luck, and that is why the greatest and dearest thank goes to Katia. She gave me the most valuable and caring support, and offered me with a precious insight in a different cultural background making me broaden my mind, but, above all, she gives me the chance to share my days with the most beautiful, sweet and stimulating creature I have ever met.
## Index of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Mob\textsubscript{adtl}: a pictorial representation</td>
<td>43</td>
</tr>
<tr>
<td>2</td>
<td>Mob\textsubscript{adtl} Specifications structure</td>
<td>60</td>
</tr>
<tr>
<td>3</td>
<td>Logic theories hierarchy</td>
<td>85</td>
</tr>
<tr>
<td>4</td>
<td>Model theories hierarchy</td>
<td>101</td>
</tr>
<tr>
<td>5</td>
<td>Theories hierarchy for the multicast/unicast example</td>
<td>116</td>
</tr>
<tr>
<td>6</td>
<td>Theories hierarchy for the explicit naming example</td>
<td>118</td>
</tr>
<tr>
<td>7</td>
<td>Integration of Mob\textsubscript{adtl} and service discovery mechanisms</td>
<td>122</td>
</tr>
<tr>
<td>8</td>
<td>Theories hierarchy for the services example</td>
<td>122</td>
</tr>
<tr>
<td>9</td>
<td>Theories hierarchy for the mobile devices example</td>
<td>128</td>
</tr>
<tr>
<td>10</td>
<td>Theories hierarchy for the migrating agents example</td>
<td>130</td>
</tr>
<tr>
<td>11</td>
<td>mapping between Mob\textsubscript{adtl} and KLAIM</td>
<td>136</td>
</tr>
<tr>
<td>12</td>
<td>Agents mobility in KLAIM implementation of Mob\textsubscript{adtl}</td>
<td>137</td>
</tr>
</tbody>
</table>
Index of Tables

Table 1. Mob_adtl model requirements: the formalisation.......................... 64
Table 2. Mob_adtl reference model axiomatisation................................... 68
Table 3. Generic Isabelle theory structure.............................................. 77
Table 4. Rules declarations from theory PC............................................. 78
Table 5. Type declarations from theory DSL........................................... 79
Table 6. Theory DSL............................................................................. 86
Table 7. Theory dDSTL.......................................................................... 94
Table 8. Support theory for Mob_adtl Isabelle theories............................. 103
Table 9. Theory mob_adtl Ao, requirements specification for the agents...... 104
Table 10. Theory mob_adtl A1, reference model for the agents.................. 104
Table 11. Applying tactics, proof of lemmaM1.......................................... 112
Table 12. Applying tactics, proof of lemmaC2.......................................... 113
Table 13. Multicast support theory........................................................... 116
Table 14. Multicast guardians theory....................................................... 117
Table 15. Unicast coordination theory...................................................... 117
Table 16. Unicast guardians theory.......................................................... 117
Table 17. Name profile support theory...................................................... 119
Table 18. Name profile guardians theory.................................................. 119
Table 19. Services support theory............................................................ 123
Table 20. Services agents theory, level 1................................................. 124
Table 21. Services coordination theory, level 1........................................ 125
Table 22. Services guardians theory, level 1............................................ 125
Table 23. Services coordination theory, level 2........................................ 126
Table 24. Services guardians theory, level 2............................................ 126
Table 25. Mobile devices support theory.................................................. 129
Table 26. Mobile devices guardians theory.............................................. 129
Table 27. Migrating agents support theory.............................................. 131
Table 28. Migrating agents agents theory............................................... 131
Table 29. Migrating agents coordination theory...................................... 132
Table 30. Migrating agents guardians theory.......................................... 133
1 Introduction

Current distributed systems heavily rely on the classical client-server paradigm, which is still predominant even if peer-to-peer communication is gaining importance and in some cases, servers have been augmented with brokerage capabilities to facilitate the discovery of available services. In this setting, the most relevant assumption about the environment of computations is that of stability: changes are likely to be slow and events like failures and disconnections are supposed to be easily dealt with, e.g. disconnected nodes will come back eventually. The main feature of classical distributed systems is the fixed structure of the network, a network where the nodes are hosts that support the execution of processes communicating via message exchange.

Although this kind of systems still keeps a central role in coping with the development of network applications, something is changing. The social need for a new way of thinking of computer-based systems, and the mere technological feasibility of moving computing away from traditional systems towards new commodity computers are fostering a great interest in solutions that exploit some form of mobility of computational entities. As a consequence, the stability assumptions are no longer relevant, since mobile entities would be able to cope with networks that lack a fixed structure: they would know how to move among hosts (nodes of the network), possibly crossing the barriers of administrative domains, looking for a computational environment that guarantees what they need in order to accomplish their tasks. Moreover, mobile entities themselves could be the building elements of dynamic networks where topology evolves upon connection and disconnections of new (mobile) nodes. Ad hoc networks could connect independent mobile devices that interact on opportunistic bases, exploiting the connections towards other entities, if available, or continuing their internal computation otherwise.

The development of methods and tools that help to reason about the properties of software applications has always been an important objective of software engineering research in formal methods. The growing use of the Internet and the World Wide Web as primary environments for developing, distributing and running programs, and the evolution of network technologies have recently increased the need for robust techniques to state and certify properties of network applications.

The specific features of mobile applications require changes in the methods of software technology. Indeed, reasoning about such applications is subtly different from reasoning about current distributed systems. Among the causes at the roots of this difference there are:

- **stronger requirements on security**, e.g. an unknown mobile application asks to enter our administrative domain;
- **network awareness**, i.e. the capability of mobile entities of changing their behaviour to better adapt to their computational and network context;
• *mobility* itself that could make difficult to trace the steps of a computation that involves many (mobile) entities.

Although many and varied contributions recently animated the research community [Bib 1, Bib 9, Bib 11, Bib 13], there are presently no standard methods, techniques or tools to support the specification, development and certification of mobile applications. This means that, from a software engineering point of view there is a big challenging issue: the definition of structural and computational models to provide designers and implementers with conceptual and programming abstractions to master the design and development of mobile applications. We need the appropriate methodological and technological tools to cope with the development of mobility-centred applications. Such tools are a model for mobility, a logic that allows for automatic reasoning on system specifications and a methodology that prescribes how to elaborate the specifications and structure the whole specification process.

The aim of building a model of mobility is to be able to describe any system based on this paradigm, so we need a model that has the right set of abstractions and the appropriate structure to accommodate the description of even very different systems. On the other hand, a formal model can be also turned into a general framework to guide the implementation of a particular class of systems, namely those systems whose structure follows exactly the structure of the model. For this reason, besides trying to keep the model as general as possible as a conceptual tool for analyzing mobile systems, it is reasonable and useful to develop an implementation of the model, that is a programming framework derived directly from the model. Having such programming framework would ease a lot the process of moving from the specification to the implementation of a system.

The work we have done in this direction has led to the definition of the model and methodology of Mob<sub>adtl</sub> and to the realization of MaRK (Mob<sub>adtl</sub> Reasoning Kit), a tool to reason about properties of Mob<sub>adtl</sub> Systems based on the theorem prover Isabelle and on the specification logic DSTL(x). Besides, we have conducted some explorations on the implementation of the model.

This thesis is organized as follows. Section 2 defines what we mean for mobility and presents an analysis of the main forces that drive the definition of a model for mobility; a survey of the most relevant models from the literature and of their applications and derivatives is given. Section 3 focuses on an analysis of extent and scope of the use of formal methods in software engineering and uses it to justify the choices done in the methodology presented in this thesis. Section 4 contains the presentation of Mob<sub>adtl</sub> describing its model, its methodology and the logic used to axiomatise the model together with its axiomatisation and the properties satisfied by the model; connections with the themes and the models presented in Section 2 are highlighted. Section 5 deals with the verification of Mob<sub>adtl</sub> systems and presents MaRK, the tool developed to this aim. Section 6 shows how to apply the proposed methodology to several examples where different issues and kinds of mobility are taken into account. Section 7 presents some explorations we have done on the implementation of the model.
2 Mobility

Mobility is the capability of an entity to change its location. Recently this word has become common in computer science, and two different forms of mobility have been defined: physical mobility refers to a scenario where hosts, i.e. computational environments, may change location in a physical space; logical mobility refers to scenarios where code, data or a combination of both may change location among the hosts of a network. As stated in [Bib 1], “Logical mobility opens up a broad range of new design opportunities, physical mobility forces consideration of an entirely new set of technical constraints, and the integration of the two is an important juncture in the evolution of software engineering as a field”. For this reason, we focus our attention on application examples and models that bring together the two kinds of mobility, and propose their integration as a means to meet the requirements posed by the users and the context.

2.1 Applicability scenarios and main issues

Mobility, both logical and physical, can be successfully exploited in several scenarios where the particular features of the context pose some difficulties to standard distributed systems and suggest new solutions that well adapt to new technological trends. Current trends in computing technology include the manufacturing of increasingly smaller, although powerful, and more portable computing devices: notebook computers, PDAs, mobile phones equipped with limited Internet access etc. These devices are meant to be used for tasks that do not require too much dependence on the environment, they are supposed to be used in a disconnected mode with regard to some base station which, when connected, provides the information that will be elaborated autonomously by the owner of the device once disconnected. On reconnection, update conflicts may be resolved automatically. The owners of these mobile devices can send mobile agents to search a network for information, disconnect from the network and collect the agents back upon reconnection.

The following scenario is an example where the integration of logical and physical mobility would offer flexibility and interoperability: the aim is to support the members of a scientific conference Program Committee (PC) during the final meeting, by keeping up to date the global state of the paper acceptance process. In our near future scenario, the PC members convene to discuss the submitted papers, each carrying the assigned reviews on a portable device, like a PDA, portable computer, etc. We assume that these devices are equipped with wireless technology, and that during the meeting, they build an ad hoc network. The application should support the integration of shared information on this ad hoc network. Besides, the hosting institution guarantees some form of access to the web, to allow PC members that are late or cannot join the meeting physically, to take part to some extent in the review remotely, e.g. at least by contributing their reviews. The purpose of the application is to enable the PC members to share and update their reviews, as well as the paper assessments as they result from the discussions. When the participants reach an agreement on the acceptance/rejection of a
paper, the shared information is updated: reviews and ratings can be adjusted, if needed, only by the reviewer, who is thus guaranteed of the fairness of the process. In the simplest situation, each member keeps the data on the device of property; when connected to the network, the local application publishes the reviews and gets access to the information already available on the network. However, we must assume that PC members can join or leave the meeting at any time, either because they are late or have early planes, or because their battery goes low and there are not power supplies available in the room. Therefore, some devices will have to provide computational power for the applications of the disconnecting members. Since the disconnection of a device could affect the overall state of the discussion, we require that members announce their disconnection. Besides, some device will have to behave as bridges towards those members that are participating remotely. As a design requirement, we use logical mobility to keep the distributed state of the discussion as complete as possible, even in this physical mobility scenario. Applications should try to migrate towards another device in order to survive to the disconnection of their computational environment from the network. The precise definition of the constraints on these migrations is left to the detailed description of the application and of the infrastructure. At our level of abstraction, we assume that some parameters are defined that describe the needs of the application, in terms of computational power, operating system, identity of the owner of the target device, etc. The support uses these parameters to manage migration requests, finding a destination to the disconnecting application that suitably distributes the load on the devices in the network. We assume there are enough resources to allow the component to migrate. In its complexity, this application offers the chance to address many challenging issues:

- the specification of applications where locations, connectivity and mobility are first order citizens, since we require the goal of the application to be reached by setting up an ad hoc network among the PC members attending the meeting, with a bridge to the web to allow late members to send in their reviews, and to provide some form of participation to members that cannot attend;
- the integration of logical and physical mobility, since we require the application to exploit logical mobility to keep the data consistent and available to the committee, even in front of the announced disconnection of a PC member;
- the verification of non functional (QoS) properties, since we require the author of a review to keep the property of the review even when it is no longer on his device, so that only he can perform or authorize changes to the review, if they are needed as a consequence of the discussion;
- the structuring of the specification:
  o to permit independent but compositional proofs of the properties of interest, like the availability of the data of a disconnected member, or his ownership of the same data. A good structure of the specification can lead to the identification of useful patterns, proved once for all, that can be applied more in
general, in situations where a shared base of information has to be kept up to date in an ad hoc environment;

- to define a clean interface between the application level and the supporting middleware. A particular care should be reserved to the presentation at the application level of the relevant events, which originate in the lower levels, like physical connections, disconnections, reconfigurations, and to the presentation to the middleware of the relevant application requests, like enquires on the network state, mobility requests, and disconnection announcements.

Applications like the one describe should guide the research activities on mobility towards an integrated treatment of physical and logical mobility. Moreover, flexibility and a seamless integration of mobility with already existing technologies should be taken in the proper consideration to provide designers with models and conceptual and technological tools for mobility. Section 6 presents several examples to show how it is possible to fulfill these requirements.

The literature offers several examples of models for mobility, logical or physical; the next section gives a taxonomic view of the main features of these models and presents three of them, the ones that mainly influenced the definition of the model proposed in this thesis.
2.2 Models for mobility: a survey

Trying to depict a general approach to the analysis of the models that deal with the description of mobile entities, the following taxonomy comes out that can be used to classify the main features of these models:

The nature of mobile entities. There is a first natural distinction between physical mobility and logical mobility: in the former, the mobile entities are computational devices that move in a physical space carrying with them the processes they host; in the latter, what is moving is a computation, or just code or data, and the mobility is from an execution environment to another one. Logical mobility deserves a further analysis since mobile entities can be any combination of code, data and control [Bib 2, Bib 3]. Models where only pieces of code can be moved support a weak form of mobility, while models where the units of mobility are processes (code + control) or agents (code + control + data) support strong mobility. Some programming languages are designed only to provide the ability of downloading code for execution (e.g. Java [Bib 4]), others support migration of entire computations (e.g. Telescript [Bib 5]). A number of distributed process calculi have been proposed as formal models for logical mobility. Among these calculi, the Distributed Join-Calculus [Bib 6], the Distributed $\pi$-Calculus [Bib 7, Bib 8], the Ambient Calculus [Bib 9], and the Seal Calculus [Bib 10] advocate programming models, which support strong mobility. Coordination-based models of behaviours are at the bases of KLAIM [Bib 11] and Mobile Unity [Bib 12].

Mobility extent. If not all the entities can move, it is useful to distinguish between mobile and stationary entities. In the Aglets API [Bib 13], the aglet context provides a bounded environment where mobile entities live. Aglet contexts are not transferable. Similarly, Telescript’s places are stationary entities. The dichotomy between stationary and mobile entities also emerges in the foundational calculi. For instance, KLAIM’s nodes and Distributed $\pi$-Calculus allocated threads are stationary entities. In the Ambient calculus, instead, ambients are the units of movement and they can be always moved as a whole including sub-ambients.

Location awareness. Location awareness results in the ability of choosing the course of action depending on the current location. The entities can be either location aware or not. The idea of location can be coupled with that of administrative domains, thus putting computations at a certain location under the control of a specific authority. In all models, the entities are location aware. The notions of ambients in the Ambient calculus, of seals in the Seal calculus, and localities in the Distributed Join calculus and Distributed $\pi$-Calculus correspond to variants of the general notion of locations. In a more programming oriented formalism like Mobile Unity, location awareness is modelled by the introduction of a variable whose value represents the current location.
Location control. The mobile entities can control their location (proactive or subjective mobility), or can be moved by other entities (reactive or objective mobility). Mobile Unity and KLAİM allow only a proactive form of mobility, while in the Seal calculus the seals are moved by their parents. The Ambient calculus is a hybrid: ambients can decide to move, but they carry with them their sub-ambients, which are thus moved in an objective way.

Communication model. There are many ways of modelling communications between mobile entities; examples are the transient shared memory of Mobile Unity, the name passing over a named channel in the Distributed Join calculus, the anonymous asynchronous message passing via explicit addressing of KLAİM. In general, remote interactions are handled through explicit naming: a component that interacts over a non-local channel has to know the place where the channel is located. An exception to this schema is the Ambients calculus: the knowledge of an ambient name is not enough to access its services; it is necessary to know the route to the ambient. Finally, interposition mechanism, wrappers that encapsulate components to control and monitor communications, have been exploited in [Bib 14]. Wrappers support the enforcement of security properties by constraining communications between trusted and untrusted components.

From the initial contribution of π-calculus [Bib 15], the first formal model of concurrency referring to mobility, several models have been proposed to cope with mobility. In π-calculus there is no formal concept of space. Mobility is equated to the ability to “express processes which have changing structure.” Under this definition any model able to pass processes or link names as values (as in π-calculus) qualifies. Models that are more recent tend to stress the importance of dealing with space explicitly and give some account of a spatial structure that is represented both in the specification language and in the underlying logic. Mob_adtl follows this trend since location and location awareness play a relevant role both in the specification logic, since spatial modalities are used to describe the state of components, and in the model where mobile entities live in a world of interconnected localities and must deal with local security policies. Mob_adtl owns much to the contributions of many researcher that have been both a source of inspiration and critical comparison. The next three sections present, together with their applications and derivatives, the models that had some influences on the definition of the Mob_adtl model: Mobile UNITY, Mobile Ambients, and KLAİM. Some of these models directly inspired the model of Mob_adtl, others provided an insight on possible choices for the communication model, and others served as a reference for the treatment of security or gave the possibility to reason about different forms of mobility. For a detailed comparison between these models and Mob_adtl see Section 4.5.
2.2.1 Mobile UNITY

Peter J. McCann and Gruja-Catalin Roman proposed this model [Bib 12] as “an extension to UNITY to address the problems of modelling dynamically reconfiguring distributed systems”. Their aim was that of extending the UNITY model [Bib 16] to describe distributed systems to directly model reconfigurations and disconnections so to cope with the issues raised by mobile computing: decoupling, context dependence, location transparency/awareness.

Following a minimalist philosophy, Chandy and Misra, developed UNITY by employing a small set of concepts, a simple notation and a proof system based on a restriction of the linear temporal logic. Their aim was to focus on the essence of concurrency rather than notational artefacts; they were looking for the "unity in the programming task". UNITY is based on the concepts of program and system, the first being the syntactical description of a process, the unit of computation, the latter a composition of several processes that execute their programs communicating via shared variables. The computational model of UNITY relies on a fairly interleaving, unbounded, non-deterministic, iterative execution of conditional statements that modify the state (sets of variable name/value couples) of processes. A set of these statements constitutes the body of UNITY programs; one statement from this set is chosen for execution non-deterministically at each step in the computation. An assumption of weak fairness guarantees that in an infinite computation each statement is chosen for execution infinitely many times. UNITY programs can be composed to build systems using superposition and union. The composition of programs, i.e. the ways in which programs can interact through variables sharing and action synchronization, is statically defined at design time. The model is supported by a temporal logic that allows to reason about properties of UNITY systems.

UNITY was meant to be used for the description of classical distributed systems, and was built on a name-based variable sharing, and statically defined actions synchronizations. In a mobile setting this is not reasonable; in order to achieve sharing of data or, more in general, in order to achieve some form of interaction, a connection between the involved entities must be available; but connections are transient due to mobility and both sharing and synchronization need to reflect this reality. The conditions that define the availability of connections depend on the location of mobile entities, e.g. proximity within a certain range allows establishing wireless links, and the components of a mobile system must be coordinated based on these conditions, i.e. dynamically. To cope with the requirements imposed by mobility, Mobile UNITY alters the model proposed by Chandy and Misra by stressing the modularity of the design and by introducing new statements that are used to describe dynamic interactions among programs.

Mobile Unity programs differ from UNITY programs in that the name spaces of different programs are kept separate, by prefixing the name actually used to declare a variable with the name of the program where the variable has been declared. This avoid the sharing of variables with the same name, thus improving the modularity of the system design, and offers a greater degree of separation between system components, a highly desirable feature in the mobile setting. Therefore, the sharing of variables in Mobile UNITY
must be explicitly defined and based on the establishing of a proper context. Another distinguishing feature of Mobile UNITY programs is their location awareness, i.e. location is something that can be used to determine the course of action. To model location, a new variable λ is attached to each program and used, for example, in the conditions that rule the interactions among components. A Mobile UNITY program can change the value of λ and thus its location. In Mobile UNITY specifications, we distinguish between the concepts of component and system. Programs that play the role of type declarations describe the components of a system. Components are instantiations of these programs at certain locations, i.e., the λ variable of each component gets an initial value and this defines an initial distribution for the components in the system. A system is a set of components together with a specification of the way components interact thus forcing a clean separation of computation and coordination. Component interactions are described in terms of transient variable sharing and actions synchronization based on context-dependent conditions. The following is an example of a Mobile UNITY system specification.

**System S**

**Program** sender(i) at λ

... end

**Program** receiver(j) at λ

... end

**Components**

sender(i) at λ₁

receiver(j) at λ₂

Programs sender(i) and receiver(j) are templates for the components sender(1) and receiver(1) instantiated in the Components section. In the Interactions section, we find a description of how and under which conditions the two components interact: when sender(1) and receiver(1) are co-located (their λ variables have the same value), the variables sender(1).x and receiver(1).y are connected in a read-write sharing. This is an example of context dependent conditional sharing: the sharing happens in response to the proximity of two components. Mobile UNITY offers a mechanism to define engage and disengage values for shared variables. In the specification of system S, the engage and disengage values are specified such that on co-location the value of sender(1).x will be propagated immediately to receiver(1).y; on disconnection, sender(1).x and receiver(1).y remain unchanged.

The read-write sharing is just one of many constructs for variable sharing and action synchronization defined in Mobile UNITY to be used in the Interactions section. These high-level constructs are actually macros built out of several new statements available in Mobile UNITY:

- **labels**: a mechanisms by which statements can be referenced in other statements;
• *inhibitors*: a way of simulating the effect of redefining the scheduling mechanisms so to avoid executing certain statements under certain conditions;

• *transactions*: sequences of assignment statements which must be scheduled in the specified order with no other statements interleaved in between;

• *reactive statements*: a mechanism for extending the effect of individual assignment statements with an arbitrary terminating computation; the reactive statements form a program that is scheduled to execute to fixed-point after each individual assignment statement including those that appear inside a transaction. This construct is at the bases of the semantics of transient sharing and synchronization.

The proof logic used to reason about properties of Mobile UNITY systems is obtained through a small technical modification of the UNITY logic. Changes are required due to the introduction of statements that extend those of UNITY, i.e., transactions and reactive statements. All the reactive statements of a system are scheduled to execute to fixed-point as a UNITY program after the execution of any other statement. In UNITY, assertions about the computation of a program are based on the Hoare triple notation $\{p\} \triangleright q$ that reads “the execution of statement $s$ in a state that satisfies predicate $p$ will take the program into a state that satisfies predicate $q$”. In Mobile UNITY, assertions about state transitions assume the form $\{p\} \triangleright s' \{q\}$ where $s'$ is a statement augmented with the effect of executing the reactive statements; in a hypothesis-conclusion notation, the resulting inference rule for this kind of transitions has the form:

$$p \land t(s) \Rightarrow q \quad \{p \land \neg t(s)\} \triangleright \{H\} \quad H \mapsto (FP(\mathcal{R}) \land q) \text{in} \mathcal{R}$$

In the rule above, $t(s)$ holds when $s$ is inhibited and $\mapsto$ is the UNITY operator used to express liveness properties. The rule states that the augmented statement $s'$ takes the computation from a state where $p$ holds to a state where $q$ holds if the following conditions are satisfied:

• $p$ implies $q$ when $s$ is inhibited, this is because $s$ will not be executed under such circumstances;

• If $s$ is not inhibited, its execution leads to a state $H$ that guarantees that the execution of the set $\mathcal{R}$ of all reactive statements will reach a fixed-point in a state that satisfies $q$.

Another rule (omitted) captures the impact of the transaction construct. All the inference rules of the UNITY logic can still be used in Mobile UNITY proofs without modification.

### 2.2.1.1 Mobile UNITY applications

The work on Mobile UNITY led to a fertile union of theory and practice ranging from the verification of real life protocols to the study of mobile code
paradigms and to the specification and implementation of several middleware-oriented products for the development of mobile application in ad hoc networks scenarios.

2.2.1.1.1 Formalization of Mobile IP

Mobile IP is a routing protocol intended to provide mobile hosts with internet connectivity when they are away from their home subnets. The aim of this protocol is to exploit well-understood methods of routing between fixed nodes to implement the routing of packets directed at mobile hosts. This is achieved by the introduction of home and foreign agents that take care of tracing the actual address of a mobile agent. The home agent corresponds to the home node of a mobile agent, while the foreign agent is the agent that each mobile agent contacts when entering a new subnet. The foreign agent communicates its own address to the home agent of each mobile agent entering its subnet, thus allowing the home agent to forward the messages directed to the mobile agents.

Mobile UNITY has been used to formalize and verify the Mobile IP protocol in [Bib 17]. The claim of that work is that the declarative nature of Mobile UNITY contributes to the development of compositional specifications, and that location-aware communication abstractions are necessary to formalize a protocol as Mobile IP intended to be the bridge between a location-dependent model and a location-independent model.

Mobile IP is formalized by giving a Mobile UNITY program for each component involved in the protocol: the mobile, home and foreign agents, and the network itself. The Interactions section specifies the synchronization and data sharing for the components. Each component can communicate with co-located components through the shared variable ether, which models the intra-subnet communications, i.e. the communications that do not need any routing. This choice allows avoiding a commitment to any particular communication medium. On the other hand, the network component models the routing machinery of the Internet and takes care of routing messages between different subnets. Each time a message from an agent in a subnet is sent to an agent in another subnet, the network component reads the value from the ether variable of the first agents and writes it in the ether variable of the second one. This is a simplification of the much more complex routing and address resolving mechanisms that realize packets delivery in real networks, but the focus of the authors is on modeling the mobility related parts of the protocol rather than on low level details. The presence of the network component models the fixed network infrastructure that is part of the assumptions of Mobile IP; the authors claim that by getting rid of this component it would be possible to deal with routing protocols for ad-hoc networks where every component can be a router, a sender or a receiver. The real-time constraint on the bounded clock drift between a mobile agent and its home agent are considered in the Interactions section via a proper model of timing constraints.

The formalization of Mobile IP in Mobile UNITY resulted in a better comprehension of the protocol itself and of the conditions under which the correctness of Mobile IP can be guaranteed. Moreover, it served as a benchmark for the Mobile UNITY proof logic that has been demonstrated well suited for the kind of proofs carried out in this exercise.
2.2.1.1.2 Code mobility

Mobile UNITY was originally thought to cope with the specification of mobile computing scenarios, in which hosts move in a physical space changing the topology of interconnections and carrying on their tasks by exploiting the available links as they are created based on relative proximity of other hosts. Settings where hosts roam a physical space are just part of a wider picture; in fact, the life of software applications, with special regard to network applications, is characterized by another kind of mobility called logic mobility. The extent of this mobility ranges through a wide spectrum, from the weaker forms that allows only the movement of portions of code to the strongest ones for which the unit of mobility is an entire application that changes location moving its code, its data and its execution state. In the search for a common foundation for all kinds of mobility, Roman, McCann and Picco explore in [Bib 18, Bib 19, Bib 20] the convenience of using Mobile UNITY to give a model for mobile code, in all its forms.

In [Bib 18] the authors take their moves from a distributed simulation example, taken from the UNITY literature, whose hard core is the calculation of a Global Virtual Time (GVT). A series of examples is developed that ranges from a traditional client/server solution to mobile solutions exploiting several form of code mobility: Remote Evaluation (REV), Code On Demand (COD) and Mobile Agent (MA). The different mobile versions of the solution to the simulation problem are built on top of the client/server paradigm in which the server knows the local times of all the clients and communicates it on request. Making use of the modular and location-oriented features of Mobile UNITY, the authors stress the decomposition of the system by factorizing the pieces of code meant to calculate the GVT and by illustrating how this code can move from the clients towards a server supposed to execute it and then send back the result (REV), or move from a server towards the clients that need to execute it (COD). The MA solution is the most distant from the client/server paradigm since it is based on a set of clients that performs their tasks independently and on a component that knows how to calculate the GVT; the latter moves in space gathering information on the local times of the clients and communicating the GVT to co-located clients.

This preliminary work led to the development of CodeWeave, a fine-grained model of code mobility [Bib 20]. This model allows for the movement of single variables, single statements, or complex block structured programs to locations on other hosts or even at points within other code fragments. The underlying idea is that of changing potentially each variable declaration and each command in a Mobile UNITY program that can move independently. A structuring of locations is introduced so to identify a world of hosts that are the execution localities where processes live and a world of processes that are the execution localities where data and code units live. Processes and units roam from host to host and processes link units as they find some data or code their need for the completion of their tasks. Code units can execute only when contained inside a process and when all their variables are connected to the variables of some data units. The dynamic linking that allows processes to connect to units and to share units with other processes is built by exploiting the transient sharing of Mobile UNITY. The model of CodeWeave captures
the semantics of this linking so that the designer is not in charge of explicitly specifying it. The introduction of distinct localities for hosts and processes leads also to a clean definition of a scoping policy for processes and code units: processes can refer only to units that are in their same host and units can share variables only with units that are in their same process. All the semantics of this model are given in the Mobile UNITY logic that can be used to perform formal verification an CodeWeave systems.

These works show that the constructs of Mobile UNITY seem to be adequate to model mobile code paradigms. The decoupled style of this formalism helped to identify the portions of code to move treating them as independent entities, and the transient sharing matches well with the dynamic linking needed by mobile code fragments that must be executed in several operating environments. On the other hand one characteristic feature of Mobile UNITY seems to not fit the requirements of mobile code paradigms, i.e. the fixed nature of systems where all the components are identified once and for all at design time and dynamic creation or destruction are not possible. Fragments of mobile code are modeled as components and this prevents from having arbitrary many instances of them and requires dealing with returning code fragments when it would be much natural to get rid of them. A tentative solution to this problem is given in [Bib 20] where a special locality is introduced called ε to serve as an infinite capacity repository of components. Components are supposed to be available in a sufficiently large number in this repository and when a new component must be created, a component of the proper type is moved from location ε to the desired location.

2.2.1.1.3 Middleware for mobility

The exploration of coordination models and languages for mobility and the commitment a clear separation between fully decoupled component executions, led to the attempt of applying the constructs of Mobile UNITY to full formal semantic characterizations of middleware for mobility. The result of this research is LIME, a middleware used to support agent coordination in mobile ad hoc networks [Bib 21, Bib 22].

The idea underlying LIME, in a setting of mobile hosts populated by mobile agents, both mobile hosts and mobile agents can be considered as instances of the same concept of mobile component and interactions happen through the transient sharing of tuple spaces accessed using the standard primitives offered by Linda [Bib 29]. LIME (Linda in a Mobile Environment) builds on the philosophy of Linda extending it to deal with mobility. The context of execution, which in Linda is a persistent globally shared tuple space, in LIME is the sharing of the tuple spaces carried by each mobile component. Each mobile component is equipped with at least one tuple space that contains the data the component wants to share with other components. On co-location, the tuple spaces of the mobile components are modified in a way that each component can access through its own tuple spaces all the tuple spaces of the components that have its same location. A first kind of sharing occurs between agents that are in the same host and defines a host level tuple space. The sharing induced by proximity of mobile hosts involves the host level tuple spaces and builds what is called a federated tuple space.
The content of a host level tuple space is dynamically updated on the bases of agents’ movements; this means that if an agent A puts a tuple t in its tuple space when it’s co-located with an agent B, the agent B will be able to see t only until the next move of A when all the tuples contained in the tuple space of A will be removed from the host level tuple space of which the tuple space of B is part. In such a situation, in order for B to use the tuples of A, a certain extent of synchronization must be introduced between the two agents. To avoid this and recover the spatial and temporal decoupling of Linda, the authors of LIME introduce a set of located primitives that allow an agent to insert tuples directly in the tuple space of a co-located agent: like hosts are localities for agents, agents become localities for their tuples. Similar operations are offered to read or withdraw remote tuples; in this case it is possible to use a host name to specify a whole federated tuple space in which to look for the needed tuple.

The merging of tuple spaces is realized by a mechanism that is based on the idea of reactive statement and transient sharing in Mobile UNITY, i.e. proper engage and disengage values are specified and the merging is the result of a single atomic transaction. Similar mechanisms are at the bases of the located operations for writing and reading tuples. Moreover, the reactive statement construct of Mobile UNITY has found a direct implementation in the reactive statement of LIME that inherits the same semantics and introduces a reactive programming style.

The development of LIME on the one hand has demonstrated the feasibility of several ideas coming from the Mobile UNITY experiences, but, on the other hand, has revealed the subtleties and difficulties that arise when trying to give an implementation to high level constructs that impose requirements on the context that sometimes may be too strict. This is particularly true for the atomicity required for engagement and disengagement operations in the transient sharing and for the assumption of the availability of routing mechanisms for located operations. The former could be impossible to guarantee in a physically mobile setting where the movements of hosts cannot be constrained and disciplined to meet the needs of the middleware. The latter is still an open issue in ad hoc networks. An implementation of LIME based on IBM’s T-Spaces overcomes these problems assuming full connectivity and announced disconnections, assumptions that can be acceptable in a small community of users carrying PDAs but not in a more general setting.
2.2.2 Mobile Ambients

Cardelli and Gordon approached the task of defining a model for mobility with the aim of describing both logical and physical mobility, or, as they call them, *mobile computation* and *mobile computing* [Bib 6]. They talk about *agents*, as the unit of computation, and *ambients* as the places where computations happen. Mobility is something that involves both agents and ambients, both computations and computational environments. On one hand, the authors of Mobile Ambients think of mobility as a way of flexibly managing some features of WANs as latency and bandwidth fluctuations. On the other hand mobility of running environments (an ensemble of data and running applications) is looked at as a promising and appealing thing to come in the near future, when we’ll be able to “transfer a piece of the desktop environment (for example, a forgotten open document along with its editor) to the laptop over a phone line”.

Taking for granted that mobility (of computations) will be soon feasible, that some technical agreement or standard will be reached to define applications able to cross the boundaries of different platforms, Cardelli and Gordon point out that mobility per se is not the main difficulty with mobile computation. In fact, all the investigation on the definition of their model is driven by the existence of *administrative domains*, abstracted in the concept of ambient. “Firewalls partition the Internet into administrative domains that are isolated from each other except for rigidly controlled pathways. System administrators enforce policies about what can move through firewalls and how”, that is what mobile agents must be able to deal with if they want to roam around following their needs. Entering an administrative domain is more than simply accessing information, more complex authorization patterns and security issues are to be taken into account since an agent must be allowed to exit from its current ambient before even asking of being allowed to enter a new one. That is why a model for mobility must capture the notions of location, mobility and authorization to move as a cohesive and interdependent set of concepts.

The leit-motif in the model proposed by Cardelli and Gordon is the concept of ambient, an entity with the following main characteristics:

- It is a *bounded* place where computation and communications happen. In fact, the computations are the content of an ambient. The boundaries of ambients determines the extent of mobility, i.e. what exactly is going to move, and the effects of mobility, in fact, to be able to move from an ambient to another one a mobile entity must cross at least the boundaries of the directly involved ambients. Communications can happen only between computations that are inside the same ambient; this means remote communications (communications crossing boundaries) are not allowed and must be properly coded in terms of computations travelling the topology of ambients carrying with them the messages to be delivered;
- An ambient can be moved as a whole. Ambients, or better their contents, model the set of data and running applications in the case
of physical mobility, and the data of a running application in the case of logical mobility. In both cases the ambient is the unit of mobility;

- It can be nested within other ambients. This choice allows easily describing the hierarchical organization that is often related to administrative domains. Ambients that are contained inside another ambient are called its sub-ambients;
- Computations inside an ambient can interact with the ambient instructing it to move or modifying the structure of its sub-ambients.

In all this, ambient names play a relevant role. In particular, names are used to refer ambients and to create capabilities, i.e. what enables a computation to act on an ambient in terms of access (entry and exit), creation, destruction and communications. The computations can create names and communicate names to other ambients. This implies that security concerns in the specification of Mobile Ambients systems are strictly connected to names and their control.

A process-algebraic calculus is used to describe ambients and their content of active computations. The description of an ambient has a form like the one of the following term:

$$n[P_1 | ... | P_p | m_1[...J | ... | m_q[...J]$$

where $n, m_1, ..., m_q$ are the ambients names, $n$ being the outermost ambient and $m_1$... $m_q$ its sub-ambients, and $P_1, ..., P_p$ are the active, i.e. running, computations contained inside ambient $n$.

The description of computations is based on some classical primitives from process algebras like action-prefix, names restriction (to model names creation), replication and parallel composition. Input/output primitives take the form shown in the following term, which represents a couple of processes running in parallel inside the same ambient $n$

$$n[(x).P | <M>]$$

The first process executes an input action which will bound the value of the free occurrences of $x$ in $P$, the second executes an output of the value $M$. This set of primitives is enriched with the following class of actions:

- in $n$
- out $n$
- open $n$

These three classes respectively model the capabilities needed to enter an ambient $n$, to exit from an ambient $n$ and to open an ambient $n$. Capabilities are built from ambient names but are not sufficient to deduce an ambient name. While the first two capabilities are fairly intuitive, the last one deserves particular care since opening an ambient means to unleash all its contained active computation in its super-ambient. This could be dangerous because the super-ambient does not know anything on the content of its sub-
ambients. On the other hand, opening of ambients is essential to implement remote communications in terms of ambients travelling the net carrying within the message to be delivered and opening themselves to release the message inside the proper ambient, i.e. remote communications come out from the interplay of mobility and local communications.

Cardelli and Gordon base the semantics for this calculus on a structural congruence and a reduction relation, in the style of $\pi$-calculus. Here only the reduction rules for the capabilities are reported so to ease the comprehension of the effects that this actions have on a net of ambients. These rules say that computations control ambients and can:

- move the ambient they are contained in and make it enter another ambient that is at its same level, like in

\[ n[in \ m. \ P \mid Q] | m[R] \to m[n[P \mid Q] \mid R] \]

where the ambient $n$ moves inside the ambient $m$;
- move the ambient they are contain in and make it exit from it super-ambient, like in

\[ m[n[out \ m. \ P \mid Q] \mid R] \to n[P \mid Q] | m[R] \]

where the ambient $n$ exits from its super-ambient $m$;
- open an ambient unleashing its content, like in

\[ open \ n. \ P \ | \ n[Q] \to P \ | \ Q \]

where the boundaries of the ambient $n$, and consequently the ambient itself, are dissolved.

2.2.2.1 Mobile Ambients dialects

Many results came out from the original work of Cardelli and Gordon to better tune the Ambient Calculus and meet the requirements of distributed mobile systems. The following sections briefly describe two of the numerous Ambient dialects, Safe Ambients and Boxed Ambients that solve some problems inherent to the original formalism and propose a better solution to security concerns.

2.2.2.1.1 Safe Ambients

Levi and Sangiorgi in [Bib 23, Bib 24] present Safe Ambients as a variation to Mobile Ambients that avoids some forms of interferences that can have place in Mobile Ambients. Interferences occur when the activity of a process is somehow disturbed (damaged, corrupted) by what another process is doing and are one of the most tricky and subtle phenomena in concurrency because they make programming and reasoning on programs pretty hard.

The authors of Safe Ambients propose some examples to explain what form interferences take in Mobile Ambients and to classify them in plain and grave interferences. The following Mobile Ambients terms

1. \[ n[in \ m.P] | m[Q] | m[R] \]
2. \[ open \ n.P \ | \ open \ n.Q \ | \ n[R] \]
are examples of what is called plain interference: an interference that happens when a process ready to perform an action can interact with more than one partner. This means that whatever the partner, not all the interactions that were still possible before the execution of the action will be possible once the action is executed. This kind of interferences is not always negative and sometimes it is useful to have such behaviour, for example to model non-determinism. The \( \text{in} \) \( m \) capability reduction and the \( \text{open} \) \( n \) capability reduction, respectively in 1 and 2, can lead to the results

\[
1.1. \quad m[Q | n[P]] | m[R] \quad 2.1. \quad P | \text{open} \ n.Q | R \\
1.2. \quad m[Q] | m[R | n[P]] \quad 2.2. \quad \text{open} \ n.P | Q | R
\]

Plain interferences are common in CCS or \( \pi \)-calculus that exhibits only this form of interferences. Mobile Ambients is also characterized by another and more dangerous form of interference like show in the following examples

\[
3. \quad \text{open} \ n | n[\text{in} \ m.P] | m[Q] \\
4. \quad h[n[\text{in} \ m.P | \text{out} \ h.R] | m[Q]] \\
5. \quad h[P] | n[\text{in} \ h | m[\text{out} \ n.Q]]
\]

In the three systems above, we have again a situation in which more than one reduction is possible but while in the first examples the possible reductions are of the same type, here the actions involved are logically different: depending on the chosen reduction, the results can vary drastically. If we focus our attention on ambient \( n \) we can see that:

- in 3 it can be opened or avoid the destruction by entering \( m \), thus giving the following two possible results

\[
3.1. \quad m[Q | P] \quad 3.2 \quad \text{open} \ n | m[Q | n[P]]
\]

- in 4 it will enter a sub-ambient of \( h \) or it will exit from \( h \), thus giving the following two possible results

\[
4.1. \quad h[m[n[P | \text{out} \ h.R] | Q]] \quad 4.2 \quad h[m[Q]] | n[\text{in} \ m.P | R]
\]

- in 5 it can let \( m \) entering \( h \) or going itself inside \( h \), thus giving the following two possible results

\[
5.1. \quad h[P | n[] | m[Q]] \quad 5.2 \quad h[P | n[]] | m[Q]
\]

To avoid such phenomena, Levi and Sangiorgi propose a variation of Ambient Calculus that introduces the concepts of co-action. The idea is that while in Mobile Ambients ambient are subdued to the decision of processes that posses the right capabilities to interact with them, they should be given the chance to refuse an interaction. Following this idea, the main difference between Safe Ambients and the calculus of Cardelli and Gordon is the presence in the former of new syntactical elements together with the proper reductions that describe the new kind of interactions. The new elements are
the co-actions \textit{in} \( n \), \textit{out} \( n \) and \textit{open} \( m \) that rules the interactions reductions as in the following

\begin{itemize}
  \item \( n[\text{in } m.P \mid Q] \mid m[\text{in } m.R \mid S] \rightarrow m[n[P \mid Q] \mid R \mid S] \)
  \item \( m[n[\text{out } m.P \mid Q] \mid R \mid S \mid \text{out } m] \rightarrow n[P \mid Q] \mid m[R \mid S] \)
  \item \( \text{open } m.P \mid m[\text{open } m.R \mid S] \rightarrow P \mid R \mid S \)
\end{itemize}

Using these new primitives and a proper type system Levi and Sangiorgi successfully recode several examples from Mobile Ambients papers that where affected by interferences and which correctness was heavily dependent on strong assumptions on the execution context. Moreover, while the algebraic theories for Mobile Ambients are poor and difficult to develop because of the presence of interferences, it is possible to develop an algebraic theory of Safe Ambients that can be used to prove, purely algebraically, correctness properties for Safe Ambients systems.

\textbf{2.2.2.1.2 Boxed Ambients}

Boxed ambients is presented in [Bib 25] by Bugliesi, Castagna and Crafa as a variant of Mobile Ambients that inherits the \textit{in} and \textit{out} capabilities with their semantics but relies on a different model of communication. Sharing the general motivations with [Bib 26] and inspired by Castagna and Vitek’s \textit{Seal Calculus} [Bib 27], the authors of Boxed Ambients depart from Mobile Ambients dropping the \textit{open} capability and introducing new primitives for directed form of communications involving an ambient and its children. The new communication primitives fit nicely the design principles of Mobile Ambients and complement the existing constructs for ambient mobility with finer-grained, and more effective, mechanisms for ambient interaction that have relevant consequences on the treatment of security and of access control in particular.

As already shown (see Section 2.2.2), opening is essential to communications in Mobile Ambients but it can be pretty dangerous to offer such a feature to processes roaming the net. In fact, ambients that carrying messages and release them by means of an \textit{open} action could host some malicious process that once unleashed inside an ambient could freely mess up the resources of that ambient interfering with other processes’ actions. Moreover, ambient interaction should be controlled by finer-grained policies to prevent from unrestricted resource access while still providing effective communication primitives. These motivations made Bugliesi, Castagna and Crafa to introduce new primitives for communication across ambient boundaries, between parent and children. In the following the new primitives together with their reduction rules

\begin{itemize}
  \item \( (\text{input } n)(x)^n.P \mid n[\text{<M>}.Q \mid R] \rightarrow P[x := M] \mid n[Q \mid R] \)
  \item \( (\text{input } \uparrow) <M>\cdot P \mid n[(x)^\cdot.Q \mid R] \rightarrow P \mid n[Q[x := M] \mid R] \)
  \item \( (\text{output } n) <M>^n.P \mid n[(x).Q \mid R] \rightarrow P \mid n[Q[x := M] \mid R] \)
  \item \( (\text{output } \uparrow)(x).P \mid n[<M>^\cdot.Q \mid R] \rightarrow P[x := M] \mid n[Q \mid R] \)
\end{itemize}

Proper tags specify the \textit{location} where the communication has to take place: for instance, \( (x)^n.P \) indicates an input from child ambient \( n \), while \( <M>^\cdot \) is an
output to the parent ambient. Standard local communication is available as well.

Directed (towards parent or child) output captures the notion of write access to a resource by identifying the ambient that is the destination of the request. Dually, directed input captures the notion of read access. Based on these intuitions, the definition of reduction enables the study of properties related to standard control policies for resource access. From Seal Calculus Boxed Ambients also inherit the two principles of mediation and locality. Mediation means that remote communication is not possible if it implies the crossing of more than one boundary, indeed it requires either mobility, or intervention by the ambients’ parent. Locality means that communication resources are local to ambients, and message exchanges result from explicit read and write requests on those resources. This allows the formalisation in Boxed Ambients of classical security policies for resource protection and access control.

One of the motivations for Boxed Ambients is to enhance static reasoning on ambient and process behavior, by enabling focused and precise analyses while preserving the expressive power of the original Mobile Ambients calculus. The results in this direction have been pretty relevant, since the new primitives actually ease the design of type systems providing precise accounts of ambient behaviour and more flexibility for communication and mobility. An interesting characteristic of these type systems is that they exhibit a rather simple structure of types that are defined as two-place constructors describing the types of exchanges that may take place locally, and with the enclosing context.
2.2.3 KLAIM

KLAIM (Kernel Language for Agents Interaction and Mobility) [Bib 28] is a language for describing mobile agents and their interaction strategies seen as a means to overcome the difficulties posed by WAN programming. KLAIM primitives and the use of explicit localities enable the programmer to distribute and retrieve data and processes to and from the nodes of a net. Moreover, localities are as first-order data and can be dynamically created and communicated over the network.

KLAIM fosters the distinction between logical distribution of processes on a net and the physical mapping of this logical distribution on the real structure of the nodes. Nodes in a network can be identified and referred to using two kinds of addresses: sites are identifiers through which nodes can be uniquely identified in a net; localities are symbolic names for nodes (processes use the reserved locality self to refer their own execution node). Sites have an absolute meaning and are alike IP addresses, while localities have a relative meaning depending on the node where they are interpreted and are like aliases for network resources. Localities are associated to sites through allocation environments at each node. KLAIM processes communicate via input/output operations over tuple spaces distributed on the nodes of a network. These operations include removing, reading or adding tuples to a tuple space and spawning processes at a node; processes can create new nodes binding them to sites. A separation of the logical distribution of processes and their physical mappings over the net leads to the sharing of the control between programmers and a net coordinator. While programmers are in charge of the design of mobile agents’ behaviour and computational facets, coordinators are in charge of the design of the net, which means dealing with the initial distribution of processes and with the setting of security and resource access policies, even mobility of processes. The coordination language is designed to handle all issues related to the physical distribution of processes. Coordinators have complete control over the network configuration changes that may be due to addition/deletion of software components and sites, or to transmission of program and of localities. The authors claim that this structuring in terms of processes and coordinators provides a clean abstraction device for global programming languages and that it can greatly help to study migratory applications and to understand the extent of configuration decisions before carrying out the actual implementation.

Process algebras and Linda [Bib 29] have influenced De Nicola, Ferrari and Pugliese in the choice of KLAIM’s primitives. Indeed, the language is an asynchronous higher-order process calculus whose basic actions are the original Linda primitives enriched with explicit information about the location of the nodes where processes and tuples are allocated. KLAIM extends Linda by handling multiple distributed tuple spaces [Bib 30]. Tuple spaces are placed on the nodes of a net. Each node contains a single tuple space and processes in execution.

At the core of KLAIM as a process calculus, lays a variant of $\pi$-calculus with process distribution and mobility, and with asynchronous communication of names through shared located repositories instead of
channel-based communication primitives. The building blocks of KLA\-IM systems are:

- **localities**, which are the addresses of nodes in a net and are the syntactic elements that express the idea of administrative domains; localities names are stored in *localities variable*;
- **nodes** and **nets**, built as follows:
  - $l::P$  
    a node, where $l$ is a locality or a locality variable and $P$ is a process;
  - $l::<l'>$  
    a located datum;
  - $N_1 || N_2$  
    the parallel composition of nets;
- **processes**: built from classical process algebraic constructs with the following located actions:
  - $out(l')@l$  
    to output the $l'$ at locality $l$;
  - $in(T)@l$  
    to retrieve from locality $l$ a datum specified by $T$, which can be a closed term or a formal parameter that will be replaced by the value actually read in the continuation of the process;
  - $eval(P)@l$  
    to spawn process $P$ at locality $l$;
  - $newloc(u)$  
    action that has the effect of creating a new node which locality will be used as the value of $u$ in the continuation of the process.

The operational semantics is given in terms of a structural congruence and of a reduction relation over nets. The structural equivalence identifies nets that represent the same net. The reduction relation gives the rules that show the evolution of KLA\-IM processes as illustrated by the following examples that convey the idea of located operations by describing the effects of an input and of a process creation:

- $l::in(u)@l'.P || l'::<l'"> \rightarrow l::P[l'/u] || l'::nil$  
  after the execution of the *in* operation, the process located at $l$ will proceed with all the free occurrences of $u$ substituted with $l''$ and the node at $l'$ will remain empty;

- $l::eval(Q)@l'.P || l'::P' \rightarrow l::P || l'::P'||Q$  
  after the execution of the *eval* operation, the process located at $l$ will proceed as described by $P$, while at the node at $l'$ the process $Q$ will be executed in parallel with the existing process $P'$.

On top of this core language, the authors build the final full fledged KLA\-IM by introducing the features of Linda, namely tuple and pattern matching. Hence, data at nodes are tuples of values, and input operations can specify patterns of values and variables to retrieve data from nodes. Moreover, an *evaluation ambient* is added to each node to resolve the localities variables used by the processes. When a new node is created, it is equipped with the
same allocation environment of the node of the process that created it. When a process is created, the result of the \textit{eval} action is to place the process at the specified node and to “link” all free variables to the local allocation environment, thus enforcing a \textit{dynamic linking} policy. On the other hand, a \textit{static linking} policy is adopted for the \textit{out} operation that first evaluates all the fields of the tuple on interest and then places it at the locality of destination. Using these two operations, two different forms of process mobility can be exploited: with the former the new process will refer to the resources as they are in the location of destination, the latter can be used to describe processes that roam a net keeping all references to the resources in the node of origin.

A logic [Bib 31, Bib 32] is given to formalize and reason about properties of KLAIM systems regarding deadlock freeness, liveness, security, resource access control and information disclosure. The proposed logic is inspired by HML [Bib 33], a temporal logic for reasoning on CCS systems. Moreover, Loreti implemented a tool [Bib 31] that allows verifying that a system satisfies a given property and that simulates a KLAIM program generating its reachability graph.

To take security issues and resource access control into account, KLAIM processes and nets are extended with type information that can be used to statically enforce security properties related to access of resources and mobility [Bib 34]. More precisely, this capability based type system allows checking whether the operations KLAIM processes intend to perform, inferred via type checking, over the sites of a net really do comply with their access rights, specified for each node by the coordinators that thus define security policies. Two variants of this basic type system are proposed to allow for a better static checking and for a finer control of process activities.

\subsection*{2.2.3.1 KLAIM’s dialects}

From the work on KLAIM stemmed several dialects of the original formalism aimed to take into account the capability of explicitly dealing with connection between nodes, higher order and highly parametric components and object-oriented features. The next sections briefly present these dialects.

\subsubsection*{2.2.3.1.1 \textbf{Open KLAIM}}

The authors presented in [Bib 35] this variation to avoid some limitations of the basic model that does not cope well with some features of open networks, the dynamic aspects of nodes creation and destruction and the changing of links. The main departure from KLAIM is that the assumption of a fixed and always available net infrastructure does not hold any more.

To this aim, the language is enriched with some new operations that allow for the dynamic updating of nodes’ allocation environments; suitable notations for explicitly expressing node connectivity are added as well, and a set of connected nodes enriches nodes besides the already existing allocation environment from standard KLAIM. Moreover, in OpenKLAIM allocation environments can be dynamically updated by executing the operation $\textit{bind}(l, u)$ that adds to the current environment the pair $(u, l)$. A new kind of processes, called \textit{NodeCoordinators}, is integrated in the existing KLAIM structure. NodeCoordinators are stationary processes and it is not possible to
use them as tuple fields. They are installed at a node either when the node is
initially configured or when the node is dynamically created performing
newloc(\(u; P\)) where \(P\) is a node coordinator. These processes can perform
ordinary KLAIM operations and coordination operations to establish new
connections, to accept connection requests and to remove connections to and
from nodes.

The result of the work on OpenKLAIM is a clean separation between
coordinator level and user level, made up respectively by NodeCoordinator
processes and standard processes. The former type of processes plays a role
similar to that of a network operating system offering to user processes the
calls to act on the network topology, or it can be seen as the part of an
application that is in charge of managing input/output connection in a
client/server paradigm.

The design principles underlying OpenKaim have been exploited in [Bib
36] to define Kaos, a calculus that can be considered as an extension of Klaim
with OpenKaim node coordinators. The main peculiarity of Kaos is that
connections among nodes are labeled by costs, special values used to model
connection features. Costs are means to program Quality of Service (QoS)
attributes at the level of WAN applications. Indeed, Kaos costs measure non-
functional properties (e.g., timely response and security) that programmers
can specify and that depend on the application. The underlying algebraic
structure of costs allows performing some desirable operations: addition,
comparison, etc. Hence, it is possible to take into account costs when paths
between nodes must be determined. Basically, Kaos mixes network
awareness, connection awareness and awareness of non-functional properties of connections.

2.2.3.1.2 Hot KLAIM

The purpose of Hot KLAIM [Bib 37] is to enhance KLAIM with features
from system F [Bib 38, Bib 39] like functional types, abstraction and
application, and polymorphic types, type abstraction and instantiation. The
aim is to provide the treatment of parameterized mobile components and to
allow for dynamic enforcing of security policies, this is possible because types
are metadata extracted at run-time and used to express trushtiness
guarantees.

HotKaim relies on the computational paradigm of KLAIM: nets are
collections of nodes, and nodes, which are multi-sets of processes and tuples
and are addressed using localities. Since process actions can be performed by
terms of any type, in HotKLAIM there is no type for processes, while there
are types for localities and tuples. The language offers the following
primitives:

- \(\text{spawn}(e)\)
  that activates a process (obtained from \(e\));
- \(\text{new}(e)\)
  that creates a new locality \(l\), activates a process (obtained from \(e\)) at
  \(l\), and returns \(l\);
- \(\text{output}(l; e)\)
  that is non-blocking and adds the value of \(e\) to the tuple space at \(l\);
- \(\text{input}(l; p \Rightarrow e)\)
that accesses the tuple space at \( l \) and fetches a value \( v \) that matches \( p \). If such a \( v \) exists, it is removed from the tuple space, and used to replace the variables \( x \) declared in \( p \) within \( e \). The operation blocks until a proper value is found.

The dynamic type checking performed in the input operation is needed because in HotKLAIM variables can have any type and there must be a guarantee for the fact that a value \( v \) matching \( p \) is consistent with the types of variables declared in the pattern.

While KLAIM provides the primitive \textit{eval} for the activation of new processes at a remote locality as a means to model asynchronous mobility, HotKLAIM comes not equipped with such an operation. In KLAIM nodes have no control on the incoming processes while in HotKLAIM mobility can happen only by mutual agreement: process abstraction can still be output at any node but these abstractions become active processes only if some process at the destination node input them. That is why higher order remote communications are needed. Moreover the \textit{eval} operation relies on dynamic scoping mechanisms that are not available in HotKLAIM that is built on a functional setting and based on parametrization. The suppression of the remote evaluation of processes permits to guarantee strong type safety properties about the evolution of net: processes can enter a node and execute only after being type-checked; this means that if a net is defined correct with regard to types, it will never give rise to type errors during its evolution. This property and the dynamic type checking define the bases for protecting hosts from unknown code and allow enacting host security at several levels.

A further extension, called MetaKLAIM, is described in [Bib 40]. MetaKLAIM supports the interleaving of computational activities with metaprogramming activities, like dynamic linking and assembling and customization of components, through the use of MetaML-style \textit{staging annotations} [Bib 41, Bib 42].

\subsection{O'KLAIM}

The authors of KLAIM worked on O'KLAIM with the intention to define an object-oriented linguistic core intended to integrate, in principle, any extension/restriction of KLAIM and also other calculi for mobility and distribution, such as Distributed Join-Calculus [Bib 6].

The design of O'KLAIM has been inspired by MoMi (Mobile Mixins) [Bib 45, Bib 46] that is the result of the integration of a basic coordination language with mixin-based object-oriented inheritance mechanisms. Standard class-based inheritance does not fit well in mobile settings, for this reason MoMi structures mobile object-oriented code exploiting the concept of mixin that is an incomplete class parameterized over a superclass. Such a choice allows for flexible solutions to problems found in mobile distributed systems like that of providing mobile agents to access local operating systems functionalities and that of preventing mobile agents from executing critical operations without control. In the former case a mobile agent can be implemented as a mixin that must be completed with a base class that contains the implementation of methods to access a particular file system. In the former, it is possible to apply to a mobile agent a mixin that redefines the methods to access sensible resources thus providing a sand-box for the
execution of the agent. MoMi defines an object-oriented calculus called Sool (Surface Object Oriented Language) that contains all the features needed to write mixin-base code. Moreover a subtyping notion is added to define class, object and mixin inheritance to be exploited at run-time. The assumption is that all code is compiled and annotated with static type information. On reception of incoming code, a type check must be performed before integrating the new code with the local one, which is supposed type correct. The received code is accepted only if subtyping-compliant with the expected one. This implies that the received code, once accepted, can be executed with no further checks on the whole code and no run-time error.

O’KLAIM integrates Sool and its subtyping within the coordination mechanism of KLAIM, much more sophisticated, complete, and effective than the one of MoMi. Syntactically speaking, O’KLAIM extend KLAIM syntax of tuples to take care of Sool object-oriented values with a special attention to typing information that are crucial in O’KLAIM. Output and input operations can be used to move object-oriented code together with standard KLAIM elements. The construct \texttt{def x = exp in P} is added in order to pass to the sub-process \( P \) the result of computing \( \text{exp} \). O’KLAIM type system does not deal with access rights, since in the new settings types are used to rule the mechanisms for the merging of code at sites. This means that there is not interest on typing the actions inside the processes: a process is well typed if it has the type \texttt{proc}. The idea underlying this approach is that all object-oriented code is supposed to be well typed and annotated with static type information produced on the home site. The formal semantics of O’KLAIM is built up from a direct extension of KLAIM’s one with a new set describing how to reduce Sool expressions to values and a new rule for the construct \texttt{def x = exp in P}. Local and global soundness results are obtained exploiting most of MoMi meta-theory itself.

\subsection{XKLAIM}

The experimental programming language X-KLAIM (eXtended Klaim) extends KLAIM by providing some programming facilities that makes easier to write actual code for real life applications. It provides a high level syntax for processes in terms of variable declarations, enriched operations, assignments, conditionals, sequential and iterative process composition.

Besides all operations from standard KLAIM, non blocking version of the input operations (\texttt{readp, inp}) are provided that act exactly like \texttt{read} and \texttt{in} do except for the fact that they do not block if a matching tuple is not found. In this case they return \texttt{false}. Moreover, also a timeout expression can be added to input operations in order to prevent processes from blocking for network latency or for the absence of the needed tuple. Since iteration is useful, a \texttt{forall} construct is included in X-KLAIM to execute an operation over each single element of a tuple space. While the use of ordinary \texttt{read} could end up in the infinitely repeated execution of the same action on exactly the same tuple, due to the non deterministic nature of the input operation, the construct \texttt{forall} guarantees that each tuple is retrieved just once. Since this iterator is not atomic, the content of the tuple space could change during the iteration, thus an explicit lock must be programmed on the tuple space if needed. Proper logical localities can be mapped to physical devices and this allows implementing I/O using tuple space operations, e.g. a screen locality.
could be mapped to the monitor and used to perform printing on the screen like in `out("Hello World")@screen`.

The X-KCLAIM reference net structure is borrowed from Open KCLAIM. This means in particular that node coordinators are included in the language and that they, and no other process, can execute all the primitives provided in Open KCLAIM to manage node connectivity. X-KCLAIM comes equipped with an object-oriented part that is based on O'KCLAIM: the syntax of X-KCLAIM processes is extended with object-oriented operations, and the syntax of method bodies is basically the same of the one of an X-klaim process. Mixins and mixins application, classes, objects, the object instantiation and the method call are included. The programmer directly writes X-KCLAIM code that is then translated by a compiler in Java code that makes use of the Java package KLAVA [Bib 43, Bib 44]. With regard to the object-oriented features, the code generated by the compiler will interact also with momi (a Java package implementing the virtual machine for MoMi). KLAVA implements the run-time systems of X-KCLAIM and can be seen both as a middleware for X-KCLAIM programs and as a Java framework for programming according to the KCLAIM paradigm. Indeed, the programmer can use directly KLAVA to exchange, through tuples, any kind of Java object, and implement a more fine-grained kind of mobility.
3 Formal methods in software engineering

The verification of mobile distributed systems is part of a wide picture that comprises the certification of functional, safety and reliability aspects of systems, with regard to human life, environmental issues and money, and the practices that, taking advantage of proper design and coding of systems, permit to limit the cost and to increase the efficiency of maintenance activities and to reuse software components to build new systems. In this picture, a prominent position is occupied by formal methods, i.e. the use of techniques from mathematical logic and discrete mathematics to describe, verify and construct systems. A formal definition of system requirements and formally defined reasoning techniques are at the bases of a comprehensive and sound description of the many facets of a system and a rigorous guide towards its realization.

Nonetheless, formal methods for a long time have been seldom adopted in industrial settings; only recently, there have been some improvements in this direction. In fact their adoption throughout the all software life cycle, or just in a particular phase, is often seen as unfeasible because too expensive in terms of personnel education, time and resources. Moreover, formal methods are mainly developed in academic settings and tested against complex but toy-sized examples that do not match well with real industrial applications often characterized by big sized specifications and by properties whose proofs relies on a high number of axioms and sub-properties. From a pragmatic point of view, the results of adopting a formalism and the related proof system can be greatly improved by the use of a tool to support the designer in the proof process at least for all those ordinary, repetitive and time-consuming tasks that do not ask for ingenuity and that often constitute most part of a proof.

The rationale behind the choices we made for Mob\textsubscript{adtl} stems from a reflection that comprises formal methods in general and their application to software design and development in real life settings. We based this reflection on some guidelines that helped us understand that it is possible and desirable to calibrate the depth and breadth of formal methods adoption and to look for a smooth integration of these methods in the everyday design activities. This section presents the general results of our investigation and the consequences it had on Mob\textsubscript{adtl}.

3.1 Extent and scope of formal methods application

The benefits of formal methods are not to be questioned, but it is true that, when thinking of introducing them into real life projects, one should take into the proper consideration the overhead that this could cause, in particular in preliminary phases when the integration is not yet completely achieved. With this intent, we tried to understand if and how it is possible to calibrate the use of formal methods with respect to the extent and the scope of their application.

In [Bib 47] Rushby presents the following four levels of degree of formalization, i.e. the extent to which it is possible to use formal methods:
0. No use of formal methods. Everything is informal, both the specification, mostly written in a natural language or in a (pseudo) programming language, with some graphics and diagrams, and the verification.

1. Use of concepts and notations from (discrete) mathematics. Some notations, derived from mathematics and logic, integrate the specifications written in natural language.

2. Use of formalized specification languages with some tool support. Proper specification languages are used and tools (syntax-driven editors, syntax checkers etc.) support these languages to a certain extent; some level of formalization is introduced in the verification process.

3. Use of fully formalized specification languages with comprehensive support environments, including mechanized theorem proving or proof checking. This level is characterized by a complete formalization of specification languages and relative proving methods, this makes possible to use automatic proof techniques.

Our aim is to define a comprehensive methodology that provides the designer with a model on which to shape its systems, a formalism to describe unambiguously their behaviour and a proof system based on the formalism that allows the designer to rigorously prove the properties of interest using a tool that brings some degree of automatization in the proof process. This approach prescribes a full commitment to formal methods as conceptual and technological tools to specify and verify a system; this qualifies our proposal to be collocated at the last level from those indicated by Rushby.

Anyway, we cannot but consider the fact that applying this degree of formalization to the entire software life cycle and to the specification of all the aspects of a system could be unfeasible since even in the presence of powerful and fully automated tools its cost could be still high. With regard to this consideration, one must carefully analyze when to adopt a formal method and choose the parts, features or aspects of the systems that must undergo a formal treatment. We turn as a source of inspiration to the analysis of the scope of the use of formal methods conducted by Kelly in [Bib 48]. Independently from the extent of the use of formal methods, Kelly depicts three different moments when formal methods can be used that are relative to two different levels in the software development activities, the management of the development process itself and the actual specification of a system:

1. Stages of the development life cycle. The formal methods are used to specify certain stages in the life cycle and to verify the steps from one phase to the next one.

2. System components. Only core components of the entire system are formally specified and verified.

3. System functionalities. Certain relevant properties of the system are verified, such as termination, safety and secrecy, deadlock freedom, etc.
We deal with systems specification and not with the formalization of the software life cycle; this excludes level 1 of Kelly’s classification. Moreover, the Mobadtl methodology imposes to describe and verify formally all the components of a system, not just some of them, and this exclude level 2. Our main interest is in the interaction and coordination of components. With regard to Kelly’s work, our position is thus that of analyzing, with respect to all the components of a system, the basic structural functionalities that allow components interaction and coordination, and to verify that the coordination satisfies the desired properties.

### 3.2 Tools to support the reasoning on systems

Among the reasons for not using formal methods, the lack of user-friendly, yet powerful tools plays a prominent role. The ideal tool would integrate seamlessly in the ordinary software life cycle process without imposing any relevant or avoidable burden on the personnel involved. Everyone should be able to use this tool in a problem-oriented way, increasing productivity and gaining more confidence, even definitive confidence, on functionalities, safety, reliability and other aspects of the system under analysis. Using formal methods, the typical operations to be performed are developing, debugging, refining and verifying specifications. Such operations imply the proof of many properties and proper tools should ease this task, possibly accomplishing it completely. The users should not be forced to gain a deep knowledge of the logic, the inference engine or the insides of the tool itself, everything that is connected with internal formal operations should be kept hidden from them.

That is in theory. The following sections briefly present three typologies of tools and underline the advantages and disadvantages deriving from using each of them, together with the reasons why we chose to base our approach on a theorem prover.

#### 3.2.1 Model Checkers

Model checkers are tools built on top of decision procedures for temporal propositional logic and are used to prove properties of finite state-transition systems. In industry, these tools are used mainly to verify hardware-centred systems or in general relatively low-level protocol or control applications.

Model checkers strength is their complete autonomy from the user. Once the specification of the system and the desired properties are given, the tool processes them without the need of external interaction and returns a result that, in case of failure on proving the properties, shows a counter example that can be of great use in debugging and redefining the specification. The use of model checkers is somehow facilitated by finite automata logics and by proper input languages, but severe drawbacks exist. Despite great efforts for the extension of model checkers to deal even with infinite domains (characterized by a finite model property), these tools are still limited in use to applications where there are not recursive functions or data structures and where, in general, the size of the state space does not explode, forcing users to split proof tasks or abstract them. Moreover, too often model checkers do not export a formal proof and this means that they do not provide certification guarantees that could be checked by third parties.
Some model checkers: SMV [Bib 49], SPIN [Bib 50], murphi [Bib 51], Java Pathfinder [Bib 53], Mona [Bib 53], the Concurrency Workbench [Bib 52, Bib 54].

### 3.2.2 Interactive Theorem Provers

Interactive Theorem provers are tools that allow to reason on logic specifications processing formulas of the specifications by applying inference rules from the logic in which the specification is written. These tools have been used to verify hardware, compilers, protocols, algorithms and program refinement and transformation.

Theorem provers strength lays in their great expressive power and flexibility. These tools allow exploiting user-defined logics (e.g. higher-order logics, temporal logics, typed logics) thus helping users on approaching the proof tasks in a problem-oriented way taking advantage from their knowledge on the application domain. Great advantages derive from the possibility of defining tactics, i.e. schemas of proofs, to be stored and reused when needed, and the possibility of storing proved theorems to be used as lemmas in other proofs. This allows building libraries of tactics and proofs that increase the applicability of theorem provers to different areas practically making these tools easily customizable and flexible. On the other hand, interactive theorem provers tend to be too often too interactive forcing users to build proofs step by step, even the smallest and most trivial ones. This means that proof time can increase significantly. Moreover, if it is good that the users of these tools can exploit their application domain knowledge to carry on proof tasks, it is not good if users are required to have such knowledge since this prevent non-experts from applying interactive theorem provers to their needs, i.e. personnel must be properly trained.

A proper choice of the application area is thus mandatory to use at their best these tools, and we believe that it is a good choice to focus our attention on architectural properties of mobile distributed systems described in our logic. No particular application domain knowledge is required, in fact, we deal with general, structural properties, and a proper set of tactics and already-proved theorems can help significantly users to cope with their tasks.

Some interactive theorem provers: NuPrl [Bib 55], PVS [Bib 56], HOL [Bib 57], Larch [Bib 58], Isabelle [Bib 59].

### 3.2.3 Automated Theorem Provers

Automated theorem provers, given a formula, search for a proof of its universal validity. Since procedures to decide validity exist for first-order predicate logic formulae, most automated theorem provers have been developed for this logic.

Automated theorem provers are rarely used in industry, despite the impressive improvements in terms of power they have undergone in the last years. There are many reasons for this lack of fortune. First of all the low expressiveness of the underlying logic, most application problems come naturally expressed as formulae in some higher-order logic. Moreover, these tools have been developed and tested using the proof search of relatively small formulae as a benchmark and they cannot cope well with real application proof tasks where the mere size of the formulae involved overwhelms their processing capabilities. The lack of a user-friendly interface
and of the capability of providing feedback if a proof cannot be found completes the drawbacks of these tools. Automated theorem provers are still at a preliminary stage. They often offer just a very powerful and sophisticated search algorithm but no more, anyway several areas of applications can be found for these tools that could “play a role in software engineering equivalent to a pocket calculator in classical engineering; simple tasks are done automatically but there is still work to be done for the engineer” [Bib 60].

Some automated theorem provers: OTTER [Bib 61, Bib 62], Gandalf [Bib 63], SPASS [Bib 64], and SETHEO [Bib 65].
4 Mobadtl: a model and a methodology for mobility

This section presents Mobadtl as the ensemble obtained by depicting a model, choosing the proper logic to formalize it and adopting a methodology that prescribes how to structure specifications and compose and refine them to obtain other specifications. The result of this activity is a conceptual and technological tool meant to help designer with the specification of distributed mobile systems by giving them the possibility to properly organize and factorize the features of their systems and to formally reason about their specifications to verify that each systems exhibits the needed properties.

At first the basic concepts on which the Mobadtl model is built are identified, then logic used to formalize the model is presented. The formalization of the model follows and it is shown how specifications of Mobadtl systems are built starting from a reference model through a refinements chain. A comparison with related works from the literature concludes this section.

4.1 Some metaphors to talk about mobility

While trying to come out with the description of such misleading and subtle entities like mobility and mobile distributed systems in general, we need to depict the right abstractions. This allows getting rid off all those details that at a preliminary stage of study may be useless or even worse dangerous as causes of loss of generality. At the very first level of abstraction, we can rightly call these abstractions metaphors, and on these metaphors, we can base our conceptual description of the general framework for the systems of interest.

Following the taxonomy presented in Section 2.2 we can start talking about what is going to move and where it is going to leave from and go. We come up with the idea of a topology of places, where some entities live and autonomously move from a place to another depending on their needs and on the connections among these places. We will call the places neighbourhoods and the mobile entities agents. Then, we will identify neighbourhoods with computational environments and agents with computational entities. We are thus speaking about logical subjective strong mobility. Physical mobility can be brought in by defining dynamically changing interconnections among the neighbourhoods to model the physical links, both wired and wireless (radio links or infrared links), between computing devices.

We have identified what is moving and where, we still have to define how these mobile entities can interact and who is going to keep them under control. Communications must be defined among agents, and for the sake of generality and for the kind of setting we want to cope with (WANs or ad hoc networks) we choose to make the agents communicate asynchronously via message passing. To take into account security concerns, we introduce for each neighbourhood just one stationary, i.e. not mobile, entity that is called guardian and that is in charge of controlling agents’ actions that involve its neighbourhood. Guardians can deny the access (in terms of message delivery
or mobility) of their neighbourhood depending on the security policies they are called to enact. Since guardians have to deal explicitly with all the actions of agents, they will be also in charge of taking care of message delivery and of the movements of agents. Details about how guardians can perform such tasks are not given, thus leaving one the chance to inject in this general picture what needed to accommodate particular scenarios. The strict bond between an agent and its guardian and the fact that a guardian practically identifies its own neighbourhood, conveys the idea of location awareness in agents. The concentration of all these responsibilities in the guardians does not violate any idea of separation of concerns, because guardians are just a conceptual tool to help factorizing all the activities that do no strictly refer to computation. Particular refinements or possible implementations of this model may still split security and routing into separate entities that would use guardians as interfaces towards the agents.

The aim of this model is to be as general as possible so to not commit to choices that can be delayed to advanced stages of the specification activity; at the same time, it provides the designer with enough handles to which to attach such future choices. That’s why it is prescribed that guardians are in charge of managing all the activities that involve their neighbourhood but nothing is said on how they are supposed to route messages and agents and which security policies they enact.

Figure 1 gives a pictorial representation of a system based on the proposed model. In the figure, we see two neighbourhoods each one with its own guardian and agents currently living in it. Plain arrows show the ideal iter that a message from an agent to another agent must follow: the sender asks its guardian to deliver the message and the guardian, interacting with other guardians, in particular with the guardian of the receiver, will actually deliver it. Dotted arrows are used to identify the analogous iter that a mobile agent must follow when willing to move. The agent asks its guardian to move to a target neighbourhood. The guardian of the moving agent will interact with other guardians to find the route to the target neighbourhood and will finally ask the guardian of this neighbourhood to accept the incoming agent.
The *cloud* in the figure stand for what is intentionally left unsaid in the model, more precisely it stands for a detailed description of the topology of neighbourhoods and for the algorithmic aspects of the inter-guardian communications. As already said, defining how neighbourhood relationships are, how guardians talk to each other and which guardians are involved in the management of communication and mobility requests is beyond the aims of this model that is meant to be just a starting point of an activity leading to the description of a real system. This activity will be defined as *refinement* in Section 4.3 and instantiated on our formalism in Section 4.4.
4.2 The logic

In the search for a logic to describe and verify global applications properties, it is natural to look at a formalism that allows reasoning on asynchronous communications and specifying separately different parts of the same applications. These requirements are characteristic of a highly distributed setting characterized by mobility and multiple security policies and DSTL (Distributed States Temporal Logic) has been defined in [Bib 67] to meet them. DSTL is an extension of temporal logic to deal with the description of distributed systems; it builds on the experience of Oikos-adtl, a specification language for distributed systems based on asynchronous communications [Bib 68, Bib 69] that has been used to formalize the first version of the Mob_adtl model. The work on DSTL has been largely driven by the requirements identified during the development of Mob_adtl, thus, while keeping its autonomy as a theoretical contribution to the field of distributed systems formalisms, DSTL finds in Mob_adtl a significant case study, both as a specification language and as verification logic. On the other hand, Mob_adtl itself has been influenced by the research on DSTL and on the specification of distributed systems. Among the results of this fertile connection, a relevant position is occupied by ΔDSTL(x), an extension of DSTL that has been developed to take advantage of the full power of First Order Logic (DSTL is based on Propositional Calculus) and to introduce the concept of event in the specification language. ΔDSTL(x) is the specification and verification language of the current version of Mob_adtl.

In the next two sections, ΔDSTL(x) is presented following the two-phased process that led to its definition. First, we introduce DSL(x), a logic thought to reason on properties of distributed states; a previous, propositional version of this logic, DSL, is presented in [Bib 70]. Then a temporal structure and the concept of event with proper operators and the relative rules are added on top of DSL(x) so to build a system apt to reason on the temporal evolution of distributed states in the absence of a global clock, a typical assumption in asynchronous settings.

The presentation of the logic is followed by a comparison with other, similar or alternative, logics.

4.2.1 Distributed States Logic: DSL(x)

The first step towards the definition of a logic that allows reasoning on the properties of the state evolution of a distributed system is to build the tools to describe the properties that the state of the systems satisfies in a particular snapshot of the overall computation. Given a set of components, we want to be able to talk about the states of each single component as well as about the relationships that hold among the states of different components like in the following:

1. \( m(p \lor q) \)
2. \( (m p \land m q) \rightarrow m q \)
where \( m \) and \( n \) are the names of two components, and the form \( mp \) has the intended meaning: \( p \) holds in \( m \). The names of the components are used to build modalities that relate properties to the system’s localities, i.e. the states of the components. What we are seeking for is a modal logic for localities that embeds the theories describing the local states of each component into a theory of the distributed states of the system. At this stage, notions like time or state transition are not considered, but, anyway, our final aim, and guiding rationale, is to define a temporal logic that allows us to write formulae like

\[
3. \quad (rn \land mq) \text{LEADS}_\text{TO} nq
\]

that reads: following a state of the system in which the local state of component \( n \) satisfies condition \( r \) and the local state of component \( m \) satisfies condition \( q \), a state of the systems will eventually be reached in which the local state of component \( n \) satisfies condition \( q \). Moreover, we want to be able to weaken and strengthen premises and consequences of formulae like 3 exploiting relations like 2. With regard to this, going back to what has been done for Oikos-adtl, in that logic it is possible to weaken the consequences of a formula like \( mp \text{LEADS}_\text{TO} nq \lor or \) including operator \( \text{LEADS}_\text{TO} \), but the rule shapes

\[
\frac{mp \text{LEADS}_\text{TO} nq \lor or}{mp \text{LEADS}_\text{TO} nq}
\]

since a formula like \( (np \land mq) \rightarrow nq \) is not part of the logic. So, the price is writing one rule for each possible weakening relation. For this reasons, while building DSL(x), we will take into consideration the implications of the available choices with regard to formulae like 3.

**Syntax.** DSL(x) is an enrichment of First Order Logic that allows reasoning on the properties satisfied locally by the components of a system. Besides First Order Logic formulae, we find formulae built using modalities that correspond to the localities represented by each component. The following is the syntax of DSL(x) formulae:

\[
F := A \mid \bot \mid \neg F \mid F \land F' \mid \forall F \mid \exists F \mid m,F
\]

where \( A \) is an atomic first order formula, \( \bot \) is the constant \textit{false}, and \( m \) is an unary location operator, i.e. a modality for a locality built using the name of component \( m_i \). With \( \overline{m_i} \) we denote the dual of \( m_i , \overline{m_i} \equiv \neg m_i, \neg F \).

With \( T \) we denote \textit{true} , \( T \equiv \neg \bot \). We let quantifiers range over modalities and \( M, N, M, ... \) are location modality variables. Binding between location variables and regular variables is possible. For example, \( \forall M.M(lam(M)) \) means that for all the components \( m_i \), \( m_i(lam(m_i)) \) holds. Quantification over modality variables is done in a standard way, following, for example, [Bib 71].
**Semantics.** To give semantics to DSL(x) we follow the Kripke’s approach of *possible worlds semantics* [Bib 72]. However, when trying to satisfy the requirements elicited so far, finding a proper frame structure is a major problem. Let’s consider a computation like the one in the following figure

\[
\begin{align*}
(n) & \quad \longrightarrow \quad \cdots \quad \longrightarrow \quad \cdots \quad \longrightarrow \quad q \quad \longrightarrow \\
(m) & \quad \longrightarrow \quad p \quad \longrightarrow \quad \cdots \quad \longrightarrow \quad \cdots \quad \longrightarrow \\
(o) & \quad \longrightarrow \quad \cdots \quad \longrightarrow \quad r \quad \longrightarrow
\end{align*}
\]

where \( m, n \) and \( o \) are the components of the system, the horizontal rows denote the sequence of states of a component and the oblique rows stand for the communications. The usual choice to build a Kripke model to give meaning to formulae like 1 and 2 in the setting of computations like the one above, would be one of the following:

1. the set of the states of a computation, i.e. the union of all the states of the system components, like the circles in the following figure.

\[
\begin{align*}
(m) & \quad \bullet \quad \longrightarrow \quad \cdots \quad \longrightarrow \quad \cdots \quad \longrightarrow \quad \bullet \\
(n) & \quad \bullet \quad \longrightarrow \quad \cdots \quad \longrightarrow \quad \bullet \quad \longrightarrow \quad \bullet \quad \longrightarrow \quad \bullet
\end{align*}
\]

This choice was adopted in Oikos-*adtl* and has shown some problems. For instance, weakening rules like 2 are not part of the logic, indeed, no world can satisfy the conjunction \((np \land mq)\) since each world refers only to one component and can satisfy only properties regarding the state of that component;

2. the set of global states, or snapshots, of the system, where each world is a tuple of states, one for each component. These tuples must satisfy some constraints to be coherent with the communications between the subsystems. In the figure below, examples of legal worlds are those of the form \(\left\{s^i_n, s^j_m\right\}\) \(i=0,1\). The couple \(\left\{s^2_m, s^1_n\right\}\) would not be a legal world because it would comprise two states belonging to two different phases of the computation separated by a communication occurred between the two components, allowing for such worlds would mean to describe the system’s evolution using uncoherent set of states.

\[
\begin{align*}
(m) & \quad s^0_m \quad \longrightarrow \quad s^1_m \quad \cdots \quad \longrightarrow \quad s^2_m \quad \cdots \quad \longrightarrow \quad s^3_m \quad \longrightarrow \\
(n) & \quad s^0_n \quad \longrightarrow \quad s^1_n \quad \cdots \quad \longrightarrow \quad s^2_n \quad \cdots \quad \longrightarrow \quad s^3_n \quad \longrightarrow
\end{align*}
\]

This choice, adopted in many logics for distributed systems, is not well suited for the case of asynchronous communication. Think of the case of property \( p \) holding only in state \( s^1_m \) and \( q \) holding only in states \( s^j_m, 0 \leq j \leq 4 \). The formula \((mp \rightarrow nq)\) would be valid in the
model, inferring a remote instantaneous knowledge that is meaningless in an asynchronous setting. Moreover, it would be natural to say that world $\langle s^2_m, s^1_n \rangle$ follows world $\langle s^1_m, s^2_n \rangle$. Going back to formulae like 3, in this case one could assert that $\forall p LEADS\_TO \forall q$ holds if $p$ and $q$ hold in $s^2_m$ and $s^2_n$, respectively, even though not even a temporal relationship exists between these two states;

3. a third possibility would be to consider all the k-uples of states (where k is the number of the system’s components) as worlds. In this case, the formula $(\forall p \rightarrow \forall q)$ would be valid in the model above if $q$ holds in all the states of component $n$. Even if this is philosophically more acceptable, we claim that a better solution can be found. Moreover, if we look at formulae like 3, this choice reveals to be wrong, indeed, if we let $p$ and $q$ hold in $s^1_m$ and $s^2_n$ respectively, we would like the computation above to be a model for $\forall p LEADS\_TO \forall q$, but world $\langle s^1_m, s^2_n \rangle$ satisfies the premise $\forall p$ and is not followed by any state that satisfies the consequence $\forall q$, thus falsifying the desiderd property.

As explained in the following, the choice for DSL(x) frames is to use the power-set of the set of all system states as the semantic domain. This choice provides us with all the desired properties and, together with an appropriate nex-state relation, makes $\Delta DSTL(x)$ expressive and able to meet the pragmatic expectations of the designer.

A frame $M$ for DSL(x) formulae is a tuple $(W, R_1, \ldots, R_k)$, where $W$ is a set of worlds that have the structure needed to give meaning to first order formulae, and the $R_i$ are the reachability relations that satisfy the following properties:

$$R1: (u, v) \in R_i \rightarrow (v, u) \in R_i$$
$$R2: (u, v) \in R_i \rightarrow (v, w) \in R_i \rightarrow v = w$$
$$R3: (u, v) \in R_i \rightarrow \forall j \neq i. \exists w. (v, w) \in R_j$$

To help the intuition, $W$ can be thought as having $k$ disjoint subsets of worlds: we call these worlds leaves. Whenever $(u, v) \in R_i$, $u$ is a leaf for relation $R_i$, namely an $i$-leaf. Condition $R1$ says that $R_i$ is reflexive on $i$-leaves; conditions $R2$ and $R3$ say that $i$-leaves are actually leaves: no other world can be reached from them. The following is an example of model, where the $i$-leaves are singleton sets, having as unique element a state of component $m_i$.
Given a model \( M \), the semantics of DSL(x) is defined as follows, where formulæ are ground:

**DSL1.** \( M \models T \)
**DSL2.** \( M \models A \iff A \) holds in all \( u \in M \)
**DSL3.** \( M \models \neg F \iff \neg M \models F \)
**DSL4.** \( M \models F \land F' \iff M \models F \) and \( M \models F' \)
**DSL5.** \( M \models m_i F \iff \exists u \in M \,(u,u) \in R_i \) and \( F \) holds in \( u \)

We propose the following axiomatization for DSL(x). For the sake of readability, we use \( m \) and \( n \), with \( m \neq n \), instead of \( m_i \) and \( m_j \). All axioms of First Order Logic hold as well.

\[
K : \overline{m}(F \rightarrow F') \rightarrow (\overline{m}F \rightarrow \overline{m}F')
\]
\[
DSL1 : \overline{m}(\overline{m}F \leftrightarrow F)
\]
\[
DSL2 : \overline{m} \overline{m} \bot
\]
\[
Nec : \overline{F}
\]

The following formulæ can be derived using DSL(x) axioms, for the proofs refer to Section 5.2.1.2 and Appendix B:

\[
axiom4 : \overline{m}F \rightarrow \overline{m} \overline{m}F
\]
\[
D1 : mmF \leftrightarrow mF
\]
\[
D2 : m(F \land F') \rightarrow (mF \land mF')
\]
\[
D3 : \overline{m}(F \rightarrow F') \rightarrow (mF \rightarrow mF')
\]
\[
D4 : \overline{m}F \rightarrow (mT \rightarrow mF)
\]
\[
D5 : \overline{m}(mF \leftrightarrow F)
\]
\[
D6 : (m(F \rightarrow F') \land \overline{m}(F' \rightarrow F'')) \rightarrow \overline{m}(F \rightarrow F'')
\]
\[
D7 : m(F \lor F') \leftrightarrow (mF \lor mF')
\]
\[
D8 : \overline{m}((mF \land mF') \rightarrow m(F \land F'))
\]
Examples. We now consider a particular class of frames. Let $S_i$ be the set of states of component $m_i$, with $S_i \cap S_j = \emptyset$ for $i \neq j$, $S = \bigcup_{i=1}^{n} S_i$, $DS = 2^S$, and $ds, ds' \in DS$. Let $(ds, ds') \in R_i$ if and only if $ds' = \{s\}$ with $s \in S_i \cap ds$. The frame $(DS, R_1, \ldots, R_n)$ satisfies rules R1, R2 and R3. Some examples follow built on these frames that are those used to build the models of $\Delta$DSTL(x).

Example 1. if we take $s \in S_m$, $s' \in S_n$ such that $p$ holds in $s$ and $q$ holds in $s'$, then, the distributed state $\{s, s'\}$ satisfies $m_p \land m_q$.

\[
\begin{align*}
(m) & \quad \Rightarrow s \\
(n) & \quad \Rightarrow s'
\end{align*}
\]

Example 2. The implication $m(F \land F') \rightarrow (mF \land mF')$ holds, while the converse does not. Indeed, for $ds = \{s, s'\} \subseteq S_m$ such that $p$ holds in $s$ and $q$ holds in $s'$, we have $ds \models m_p \land m_q$ but not $ds \models m(p \land q)$.

\[
\begin{align*}
(m) & \quad \Rightarrow s \\
(n) & \quad \Rightarrow s'
\end{align*}
\]

Example 3. The formula $m \forall F$ is false. Indeed, $ds \models m \forall F$ if and only if there exists an $s \in S_i \cap S_m \cap ds$ such that $F$ holds in $s$, but no such $s$ can exist since $S_m$ and $S_n$ are disjoint. Conversely, $m \forall F$ is satisfiable and is equivalent to $mF$.

Since DSL(x) carries over all meaningful propositional rules, like and simplification, in such a way that they can be exploited orthogonally to temporal operators, the exploitation of the local theories becomes smooth and robust, while proving distributed properties.

4.2.2 Distributed States Temporal Logic with events: $\Delta$DSTL(x)

$\Delta$DSTL(x) is an asynchronous, distributed, temporal, first order logic built on top of DSL(x). To describe the evolution of computations, we add a temporal structure to the semantics of DSL(x) and define a set of operators that relate conditions in different states of a computation. $\Delta$DSTL(x) includes also the operator $\Delta$ to specify events.

Syntax. Given a DSL(x) formula $F$, the following is the syntax of $\Delta$DSTL(x) formulae:
\[ \phi ::= \forall \exists F \mid \Delta F \mid \text{INIT } F \mid \text{STABLE } F \mid F \text{ LEADS \_ TO } F' \mid F \text{ LEADS \_ TO \_ C } F' \mid F \text{ BECAUSE } F' \mid F \text{ BECAUSE } \_ C F \]

A DSL(x) formula is a \(\Delta\text{DSTL(x)}\) formula, provided it is a closed sentence, that’s the meaning of \(\forall \exists F\). A formula shaping \(\Delta F\) is used to express the becoming true of a condition \(F\), i.e. it used to define an event. Operator \(LEADS \_ TO\) expresses a liveness conditions and is similar to Unity \(\rightarrow\) (leads to); the intended meaning is that a state of the systems that satisfies \(F\) is always followed by a state of the systems that satisfies \(F'\), i.e. \(LEADS \_ TO\) defines the sufficient causes for a condition to hold. Operator \(BECAUSE\) expresses a safety condition; the intended meaning is that a state of the system in which \(F\) is satisfied must always be preceded by a state of the systems in which \(F'\) is satisfied, i.e. \(BECAUSE\) defines the necessary causes for a condition to hold. Operator \(STABLE\) extends Unity’s \(stable\) to the distributed case and defines stability of conditions, i.e. it states that a condition, once established, will keep to stay true. Operator \(INIT\) defines the conditions that hold in the initial state of a computation. Operators \(LEADS \_ TO\) and \(BECAUSE\) can be postfixed with a \(_C\) when the condition in the consequence (right hand side operand) has to hold in a state \(close\) to the one that satisfies the premise (left hand side operand), i.e. the same state or its immediate successor/predecessor.

At this point, a short philosophical note is needed. We tend to think that our operators express \(causality\), even though, strictly speaking, they only define temporal relations, i.e. that their consequences hold after (or before, depending on the operator) their premises. In fact, in our models, a state in a component follows a state of another component only if there has been a communication between the two. Philosophically, this may not entail a causal relation, but our goal is to specify systems, and it is thus natural to think that the communication carries the information needed to cause the intended effect. It is in this sense that we use the term causality.

When not otherwise specified, a formula \(\phi\) is to be intended as universally quantified over all values of the variables appearing in the premise of \(\phi\) and existentially quantified on the remaining variables. For example, the formula

\[
M(r(x) \land q(y)) LEADSTON (r(x, M, z))
\]

is to be interpreted as a shorthand for

\[
\forall M, x, y \exists N, z M(r(x) \land q(x)) LEADSTON (r(x, M, z)).
\]
**Semantics.** The models for $\Delta$DSTL($x$) formulae are built on structures like the one in the following figure, which describes the computation of a system with two components. Here $p$, $q$, ..., are the properties holding in the states.

\[(m) \quad \circ \quad p \rightarrow q \quad r \quad \rightarrow u,z \rightarrow z \rightarrow z \rightarrow \]

\[(n) \quad \circ \quad p,t \rightarrow u \rightarrow v \rightarrow p \rightarrow w,t \rightarrow w,t \rightarrow \]

We call $S_i$ the set of states of component $m_i$, and $S$ the set of all the states of the computation. A distributed state $ds$ is any subset of $S$, and $ds^0$ is the set of the initial states. We consider also extra states, represented with $\circ$ in the figure, that precede the initial states. We assume they belong to $S$, and use them to give semantics to the events in the initial state. The following relation defines the semantics of literals in the initial state

$$\{y_j^{-1}\} \models L \text{ iff not } \{y_j^0\} \models L$$

i.e. a literal holding in an initial state does not hold in the extra state preceeding it. Once we have defined the realation between what holds in the initial state and what holds in the extra states, we can define the following relations on the temporal ordering of states

- $ds > ds'$ iff each state in $ds'$ is followed by a state in $ds$ and each state in $ds$ is preceded by a state in $ds'$.
- $ds \geq ds'$ iff each state in $ds'$ is either in $ds$ or followed by a state in $ds$, and each state in $ds$ is either in $ds'$ or preceded by a state in $ds'$.
- $ds \geq_c ds'$ iff each state in $ds'$ is either in $ds$ or immediately followed by a state in $ds$, and each state in $ds$ is either in $ds'$ or immediately preceded by a state in $ds'$.
- $ds >_c ds'$ iff $ds \geq_c ds'$ and $ds$ does not include $ds'$.
- $ds >_1 ds'$ iff all the states in $ds'$ have a local immediate successor in $ds$, and, viceversa, all the states in $ds$ have a local immediate predecessor in $ds'$.

Formally a state $s$ is a local successor of a state $s'$ iff $\{s\} >_c \{s'\}$ and $s \in S_i$ implies $s' \in S_i$. When $ds$ includes some initial states, $ds'$ includes the extra states preceeding them. Note that each distributed state has a unique local predecessor.

Now we can give meanings to events with the following

$$ds \models \Delta F \text{ iff } ds \models \text{ and } \exists ds' : ds' <_1 ds \text{ implies } ds' \models \neg F$$

Being $M$ be a model, the semantics of $\Delta$DSTL($x$) is given by the following rules, where $\models$ is defined as for DSL($x$) formulae except for the case of the $\Delta$ operator:
Recall that a distributed state is any set of states. This means that when we have to check a condition like $\forall ds \geq ds^0 \ldots \exists ds' \ldots$, we need to consider all possible subsets of $S$. This may lead to counter-intuitive choices, like taking larger states than needed. However, the specifier can be safely guided by the natural interpretation of the operators. Anyhow, our definition of distributed state is exactly what we needed to overcome the problem with the existing logics for distributed systems, which do not have the right expressive power to reason on systems’ behaviour when the communication is based on asynchronous message passing.

Axioms and rules of $\Delta$DSTL(x):

**Necessitation.**

$\text{Nec } F \frac{F_{\text{(DSL(x) closed formula)}}} {F_{\text{(DSTL(X) formula)}}}$

**Introduction and elimination.**

- **LeI** $F \text{ LEADS } TO \text{ C F}$
- **LcI** $F \text{ LEADS } TO \text{ C G}$
- **LI** $\frac{F \text{ LEADS } TO \text{ C G}} {F \text{ LEADS } TO \text{ G}}$
- **BcI** $F \text{ BECAUSE } \text{ C F}$
- **Bi** $F \text{ BECAUSE } \text{ C G}$
- **BI** $\frac{F \text{ BECAUSE } \text{ G}} {\text{INIT } \neg F}$
- **LE** $\frac{F \text{ LEADS } TO \perp} {\neg F}$
- **BE** $\frac{F \text{ BECAUSE } \perp} {\neg F}$
- **DE** $\Delta F \rightarrow F$
- **SI** $\frac{F} {\text{STABLE } F}$
- **SI** $\frac{\text{INIT } MF} {\text{STABLE } MF}$
- **SE** $\frac{F} {\neg F}$
- **MF** $\frac{F} {\neg F}$
- **BSE** $\frac{\text{STABLE } MG} {\text{STABLE } MG}$
- **ΔI** $\frac{F \text{ LEADS } TO G} {F \land \neg G \text{ LEADS } TO \Delta G}$
- **BSE** $\frac{\text{STABLE } MG} {\text{STABLE } MG}$
- $\frac{F \rightarrow G} {\neg (F \rightarrow G)}$
Transitivity.

\[
\begin{align*}
&\text{LTR} \\
&\begin{array}{c}
F \text{ LEADS } _{\text{ TO }} F' \\
F' \text{ LEADS } _{\text{ TO }} G \\
F \text{ LEADS } _{\text{ TO }} G
\end{array} & \begin{array}{c}
F \text{ BECAUSE } F' \\
F' \text{ BECAUSE } G \\
F \text{ BECAUSE } G
\end{array} \\
&\text{BTR}
\end{align*}
\]

Premises and consequences strengthening and weakening. The following are rules schemas, instance rules can be obtained making \( OP \) to range over the temporal operators \( LEADS _{\text{ TO }}, LEADS _{\text{ TO C }}, BECAUSE , BECAUSE _{\text{ C }}, \) the name of rules are obtained replacing the * with L, Lc, B or Bc respectively.

\[
\begin{align*}
&\text{*SW} \quad \frac{G \rightarrow F \ F' OP F' \ F' \rightarrow G}{F OP G} \\
&\text{*PD} \quad \frac{F OP G \ F' OP G}{F \lor F' OP G} & \frac{F OP G \ F OP G'}{F OP G \land G'} \\
&\text{*CC}
\end{align*}
\]

Notification and Confluence.

\[
\begin{align*}
&\text{Nf} \quad \frac{F \text{ BECAUSE } G \ G \text{ LEADS } _{\text{ TO }} MG' \ STABLE \ MG'}{F \land M \overline{F} \text{ LEADS } _{\text{ TO }} MG'} \\
&\text{Cf} \quad \frac{STABLE \ MF \ STABLE \ MF'}{MF \land MF' \rightarrow M (F \land F')}
\end{align*}
\]

Properties of the initial state.

\[
\begin{align*}
&\text{I1} \quad \text{INIT } M \overline{F} \\
&\text{I2} \quad \frac{\text{INIT } M F \overline{F}}{\text{INIT } M F} \\
&\text{I3} \quad \frac{\text{INIT } M \overline{F}}{\text{INIT } M F} \\
&\text{IW} \quad \frac{\text{INIT } F \ F \rightarrow G}{\text{INIT } G} \\
&\text{IC} \quad \frac{\text{INIT } F \ \text{INIT } F'}{\text{INIT } F \land F'}
\end{align*}
\]

In Appendix B are reported the mechanized proofs of many interesting theorems of \( \Delta \text{STL(x)} \) that can be derived from this axiomatisation.

Examples. The following examples present a discussion on the satisfiability of some formulae with respect to the computation and some considerations about the consequences of the choice we made for the semantic domain of DSL(x).

Example 1.
\[
M \models \exists u \text{ LEADS } _{\text{ TO }} m u \land n t \\
M \models \exists u \Delta u \text{ LEADS } _{\text{ TO }} m u \land n t
\]

Consider the first formula: a distributed state satisfies the premise if it contains, for instance, the second state of component \( n \), where \( u \) holds, call it \( s \). It is immediate to find a distributed state following \( \{s\} \) and satisfying the consequence, e.g. the first pair of states where \( u \) holds in \( m \) and \( t \) holds in \( n \) (related by a communication in the figure), call this pair \( P \). Any larger
distributed state $ds$ including $s$ satisfies $m_u \land n_t$. The same considerations hold for the second formula, since $\{s\}$ satisfies $\Delta u$ as well.

$$M \models_T n \Delta w \text{LEADS}_\text{TO}_C m_u$$
not $M \not\models_T n w \text{LEADS}_\text{TO}_C m_u$

Only the $5^{\text{th}}$ state of $n$, where $w$ becomes true, is followed by a state of $m$ where property $u$ holds.

$$M \models_T n w \text{BECAUSE} n p \land m u$$
not $M \not\models_T n w \text{BECAUSE} n(p \land u)$

Consider, for instance, the first state, say $s$, of $n$ where $w$ holds. The set $\{s\}$ is preceded by the pair $P$ composed by the initial and the second state of $n$. The set $P$ satisfies the consequence of the first formula. Any superset $ds$ of $\{s\}$ is preceded by $ds \cup P$. On the contrary, to satisfy the second formula, we would need a singleton state of $n$ satisfying both $p$ and $u$.

$$M \models_T \text{STABLE} n(w \land t)$$

Any distributed state including a state of $n$ satisfying $w \land t$, is immediately followed by a distributed state including a state of $n$ satisfying $w \land t$.

$$M \models_T \exists M \overline{m}(u \rightarrow z)$$
There exists a component, $m$ in our case, each state of which satisfies $u \rightarrow z$.

$$M \models_T \text{INIT} \Delta p$$
In both initial states property $p$ holds. If a literal hold in the initial state, the corresponding event holds as well.

*Example 2.* The choice of $2^S$ as a semantic domain of the distributed state logic formulae, and the non-equivalence between $m(F \land F')$ and $mF \land mF'$ are useful to specify that a given condition can have different future effects, without constraining them to occur in the same state. Similarly, we can express complex preconditions in a temporal formula. For instance, assume we want to specify and reason on the delivery of credit cards to customers. The bank, for security reasons, sends the card and the code separately. Once the customer has got both of them, he is allowed to withdraw money from an ATM machine:

1. $\text{bank(new\_card)LEADSTOuser(receive\_card) \land user(receive\_code)}$
2. $\text{user(can\_withdraw)BECAUSEuser(receive\_card) \land user(receive\_code)}$

The equivalence between $m(F \land F')$ and $mF \land mF'$ would have required the following specification, since the formulae above, interpreted under this equivalence, would be too restrictive, asking for card and code to be received at the same time:
(3) \texttt{bank(new\_card) LEAST\_TO user(receive\_card)}
(4) \texttt{bank(new\_card) LEAST\_TO user(receive\_code)}
(5) \texttt{user(can\_withdraw) BECAUSE user(receive\_card)}
(6) \texttt{user(can\_withdraw) BECAUSE user(receive\_code)}

Last, but not least, with an eye to the first order, a formula like (1) makes it easier to bind variables in \textit{card} and \textit{code} than with the unrelated formulae (3) and (4).

\subsection{4.2.3 A comparison with related logics}

This section is dedicated to a comparison of $\Delta\text{DSTL(x)}$ with other logics that approach similar problems to model distributed systems. This comparison, based on Bib 67, should clarify the rationale behind the choices we made for our logic.

\textit{Classical Logic}. A point of discussion is why we need the modality $m$ rather than a distinguished propositional symbol $\text{here}_m$ to replace systematically each sub-formula $\text{mF}$ with $\text{here}_m \land F$. One motivation is that we do not want the equivalence between $\text{m}(F \land F')$ and $\text{mF} \land \text{mF}'$, as discussed previously, and the two translations $\text{here}_m \land F \land F'$ and $\text{here}_m \land F \land \text{here}_m \land F'$ would be equivalent. More importantly, $\text{mF} \land \text{mF}' \text{LEAST\_TO} \text{F''}$ would be translated in the formula with a false premise $\text{here}_m \land F \land \text{here}_m \land F' \text{LEAST\_TO} \text{here}_o \land F''$.

\textit{Hybrid Logic}. Hybrid logic allows the specifier to directly refer to specific points (states) in the model using \textit{nominals} [Bib 73]. A nominal $i$ is an atom that is true at exactly one point in any model. The operator $\text{@}_i$ permits to jump to the point named by nominal $i$. We might consider defining an hybrid signature including distinguished sets of state variables, one for each component, and translate $\text{mF}$ in $\exists x. \text{@}_x F$, where $x$ is a state variable in the appropriate set. Likely, the resulting setting would be more complex than that offered by $\Delta\text{DSTL(x)}$.

\textit{Metric and Layered Temporal Logic}. Some similarities can be found between our location operator and the MLTL operators defined in [Bib 74] that make it possible to compose formulae associated with different time granularities and to switch from one granularity to another. Time instants are organized in temporal domains, and the set of temporal domains is totally ordered with respect to the coarseness of the domain elements. To look for an embedding of $\Delta\text{DSTL(x)}$, we can consider three domains: \textit{system}, with a unique element; \textit{components}, whose elements are the components $m_1 \ldots m_k$; \textit{states}, the domain of the states. Then the formulae are translated using an appropriate combination of MLTL operators. For instance, the translation of $\text{mF}$ should be $\Diamond_{m} \text{\Delta_{component}} \land \exists x \text{\Delta_{state}} F$. Since the full expressive power of MLTL is likely not needed, the simpler framework of $\Delta\text{DSTL(x)}$ is of pragmatical interest.
Other logics for distributed systems. Various extensions of temporal logic have been defined in the literature to deal with distributed systems.

In TTL [Bib 75], for each local state of the system, a visibility function specifies which remote information is accessible. The visibility function is defined based on a relation among states which is symmetric in the case of states belonging to distinguished components.

A trace-based extension of linear time temporal logic, called TrPTL, has been defined in [Bib 76], see also [Bib 77]. The logic has been designed to be interpreted over infinite traces, i.e., labelled partial orders of actions, which respect some dependence relations associated to the alphabet of actions.

In [Bib 78], a temporal logic, StepTL, is defined and interpreted over multistep transition systems. These are a well-known extension of transition systems, permitting to describe as concurrent the steps of computation that can actually be executed in parallel. A multistep transition system thus contains transitions of the form $sAAs'$, where $A$ is a set of actions, instead of a single one.

Three distinguished logics are presented in [Bib 79] to describe systems composed of sets of communicating agents. The logics differentiate on the amount of information each agent can have on the other agents running on the system, but share a common setting: agents communicate via common actions. The models for these logics are runs of networks of synchronizing automata. The logics $D_0$ and $D_1$ presented in [Bib 80] are based on a similar approach.

In all these proposals, components communicate via some form of synchronization, and logic formulae are interpreted on models shaping:

$$
(n) \quad \cdots \cdots \cdots \rightarrow a \cdots \cdots \cdots \rightarrow b \cdots \cdots \cdots \rightarrow c \cdots \cdots \cdots \cdots \cdots \cdots \\
(n) \quad \cdots \rightarrow d \cdots \cdots \cdots \rightarrow e \cdots \cdots \rightarrow f \cdots \cdots \cdots \cdots \cdots \cdots
$$

Therefore, in any logic defined over these models, it is not possible to express the asymmetric nature of causality we are interested in when modelling the behaviour of agents communicating asynchronously by message passing. Indeed, in the previous model we can assert both that $ma LEADS TO nf$ and that $nd LEADS TO mc$.

A logic closer to $\Delta DSTL(x)$ is proposed in [Bib 81], where a branching time temporal logic for asynchronously communicating sequential agents (ACSAs) is defined. ACSAs communicate asynchronously via message passing. The logic contains temporal modalities indexed with a local point of view of one agent and allows an agent $i$ to refer to local properties of another agent $j$ according to the latest message received: an agent can gain information about another agent by receiving messages but not by sending them. We allow agents to make remote future assertions: therefore, it is easier to express global liveness properties.

Knowledge Logic. A logic to reason on asynchronous message passing systems is proposed in [Bib 82]. The language used, $L^U_n$, is obtained by extending their language of knowledge with the modal operators $U$ and $O$. 

57
Formulae in $L_n^U$ permit to express how the $n$ agents in a system gain knowledge over time. A set of characteristic formulae valid in the logic are presented, but a sound and complete axiom system is not defined. The authors focus their attention on systems based un-reliable communications, while only state that properties of reliable communications can be expressed. A major difference with our work relies on the models used to interpret $L_n^U$ formulae. Even if the knowledge of the agents is limited to their current local histories, i.e. sequences of messages sent or received and of internal actions, interpretation structures are based on global time and state.

**Partial Order Temporal Logics.** Partial Order Temporal Logics (POTL) [Bib 83] permits to deal with the causal relationships between the events of a set of processes executing concurrently. The Interleaving Set Temporal Logic (ISTL) [Bib 84] extends POTL with features from linear temporal logic and branching temporal logic. The Kripke structures for both logics are very different from the one defined in this paper. We are addressing a specific class of systems that we consider very relevant nowadays, which are distributed systems with asynchronous message passing. These systems have a few notable characteristics: there is no global state, and interactions among components occur only via messages. Therefore, a specification is essentially devoted to describing the causal relationships among the components. We think that these characteristics are so important that the designer working on a specification will greatly benefit if they are naturally embedded in the basic model he is using. Hence, the investigation in Kripke's structures presented here.
4.3 Refinement as a methodological choice

The Mob\textsubscript{adult} design methodology is based on the notion of refinement [Bib 85] and this section explains how we apply this concept to the structuring and developing of Mob\textsubscript{adult} specifications. The basic idea is that we give a first high level description of our systems in terms of the description of the behaviour of agents and guardians and then we incrementally add details to it thus building a chain of descriptions every one of which describes the same system as the previous ones but at a lower level of details. The activity of adding details to a specification while preserving the properties satisfied by that description is called refinement.

The formal descriptions, written in ADSTL(x), of Mob\textsubscript{adult} systems are organized in theories and the refinement relation between these theories corresponds to logical deduction: we say that a theory $T'$ refines theory $T$ if and only if any formula $F$ holding in $T$ holds also in $T'$. We call specification the set of axioms defining a theory and we structure theories so that the specification of each class of homogeneous components, i.e. agents and guardians, is kept separate from the specification of other classes, and can be refined independently, in a fully compositional way. However, complete independence would require fixing the interface of the components in each class already at the zero refinement step, where requirements are first specified. This means that the refinement steps can extend the interfaces, but not modify them. To avoid imposing this constraint to the designer, we adopt a different structuring and refinement strategy.

Let’s call $A$ and $G$ respectively the theories for the specifications of agents and guardians. The specification of a system, at any level of the refinements chain, is the composition of both the specification of the two classes, and the specification of the way they coordinate their behaviours, contained in theory $C$. The specification of a class contains only local axioms, i.e. axioms that refer to a single component of that class, or axioms relating events and conditions in different components of that class. The coordination axioms relate remote events and conditions of components in different classes.

At each refinement step it is possible to extend the set of axioms to specify new properties, or to refine and distribute the formulae in the existing theories. For instance, given agent $m_A$ and guardian $m_G$, formula $m_A p \text{LEADS}_TOM_c q$ in the coordination theory could be refined by introducing $m_A p \text{LEADS}_TOM_{a,r}$ in the theory of class $A$, $m_A r \text{LEADS}_TOM_{c,s}$ in the theory of class $A$, and $m_c s \text{LEADS}_TOM_c q$ in the theory of class $G$. To guarantee that this refinement is correct, we need to prove that the property stated by the axiom we replaced is still derivable as a theorem; this is easy to do since we can proceed as follows:

\[
\begin{align*}
\text{LTR} & \quad m_A p \text{LEADS}_TOM_{a,r} \quad m_A r \text{LEADS}_TOM_{c,s} \\
\text{LTR} & \quad m_A p \text{LEADS}_TOM_{c,s} \quad m_c s \text{LEADS}_TOM_c q \\
\text{LTR} & \quad m_A p \text{LEADS}_TOM_c q
\end{align*}
\]
The final goal of the refinement process is to specify the classes fully and
to leave in the coordination theory only the assumptions on the underlying
middleware or the properties that will have to be guaranteed by the
application context. A typical middleware property is that point-to-point
communication is reliable; assumptions on the context can include human
behaviour.

Figure 2 shows the overall refinement/extension structure of the Mob
adtl theories.

![Diagram](image)

**Figure 2. Mobadtl specifications structure.**

At the zero refinement step we define the global properties of the model.
We define the refinement relation between theories accordingly to the
proposed refinement strategy: we ask that at each step theory $A_i$ refines $A_{i-1}$,
$G_i$ refines $G_{i-1}$, and $A_i \cup G_i \cup C_i$ refines $C_{i-1}$. Indeed, the first refinement
step, corresponding to the axiomatization in Section 4.4.2, shows a first
distribution of the coordination requirements to agents and guardians. The
lower levels represent further refinements, like those of the examples in
Section 6. At each refinement step, we can derive the general properties that
hold in the system: we consider the union of all the theories at the step, and
reason on them using the rules and the theorems of the logic. The derivation
of the general properties is needed to prove that the refinement is correct.
Moreover, since the step can extend the set of axioms in each theory, some
new properties can hold in the system because of the new axioms.
4.4 The axiomatisation

This section presents the axiomatisation of Mob_adtl as the result of a formalisation in ΔDSTL(x) of the conceptual description of the model we gave in Section 4.1. At first we depict the main properties that our model is required to satisfy, i.e. the requirements of the model, and give a formalisation of them. Then we refine this first description to provide a more detailed specification of the behaviour agents and guardians must exhibit in order to guarantee the properties previously stated.

4.4.1 The requirements of the model

In this section, we first describe the requirements of the Mob_adtl model, and then we formalize them in ΔDSTL(x). To present the requirements we gather them according to the following features:

Location. In Mob_adtl, a network application consists of an immutable net of elaboration nodes: the neighbourhoods. A neighbourhood is a bounded environment where several components live: it corresponds to a logical locality and can represent a unit of physical mobility. Components model units of executing computation. Components can be stationary or mobile: we call agent a mobile component. The components can communicate, also remotely, and the agents can move, changing their neighbourhood.

Net topology. Nothing is said in the general model about the logical structure of the net. The only constraint is that neighbourhoods are not nested in a hierarchy. In the refinements, specific routing policies can implement more structured topologies, e.g. one where a neighbourhood plays the role of firewall for a set of other neighbourhoods. In this case, all the traffic is routed via the firewall.

Location awareness. Each neighbourhood is associated with a particular stationary component, the guardian. The knowledge of their own guardian makes components location aware.

Authority. Guardians monitor the components in their neighbourhood and limit the resources they can use, playing the role of authorities. Movement requests can be refused by guardians, e.g. for security reasons. More in general, the guardians intercept messages and agents and decide which of them can enter or leave the neighborhood they control. They also provide the routing facilities to forward messages and to handle moving agents: however, we require that the Mob_adtl model does not constrain routing policies a priori. The same hold for agent migration: movement requests must be examined at least by two guardians: those of the origin and target neighborhoods. We do not require that these guardians explicitly accept the request. This is done on purpose, so that any more specific policy, e.g. “agree with silence”, can be introduced in the refinements.
**Mobility.** Agents can move from neighborhood to neighborhood. The model supports strong mobility, since the units of mobility -- the agents -- are made up of code, data, and control.

**Subjective mobility.** Mobility is subjective, that is the agents can control their location and cannot be passively moved by other entities.

**Communication.** Communications are based on asynchronous message passing, and communications between components occur via the guardians.

**Naming and profiles.** Guardians and components have a unique name. The target of communication and movement requests can be described implicitly by a profile, i.e. a set of constraints. The constraints predicate on the attributes of the target, e.g. by explicitly naming the target itself. As far as mobility is regarded, the use of a profile to specify a target neighborhood enables, for instance, guardians and agents to negotiate the resources needed before committing to a movement. As far as communication is regarded, a profile can describe the receiver component in terms of security constraints and exported services (in case of a message request to a server component). As a further example, profiles permit to smoothly integrate requirements on the quality of service in the early design stages. The guardians are in charge of resolving profiles to match them against a set of components that satisfy the expressed constraints. The choice of adopting a mechanisms like the one of profiles, gives the designer a great freedom with regard to the commitment to particular forms of communication, mobility and, more in general, identification; on the other hand, the refinement based approach grants the possibility of instantiating the profile resolution mechanism to realize any particular form of component identification.

Following these guidelines, we proceed at giving a formalization of the requirements of the model, i.e. the level 0 of the refinement process towards the specification of MoBadt model. The result of this activity is reported in Table 1.

Location awareness is encoded in the components’ state via predicate guardedby(G), that is, the current guardian identifies the current location. Stationary components are characterized by the invariance of this predicate. As a consequence of S2 and S5, agents are always in some place: agents always knows where they are, i.e. the name of the guardian of the current neighbourhood, and their knowledge is updated consistently on location change. In the initial state every guardian knows the agents in its neighbourhood, and every agent knows the name of its guardian (ILA); the same property cannot be stated for the other states of the system because of the asynchronous setting, indeed the initial state is the only one where we assume a global knowledge of the system. Agents are not ubiquitous, they are always in exactly one neighbourhood (U). Guardians are not guarded, i.e. locations are not nested (N), for a consideration about how to recover structured topologies see Section 4.5.1.

Each guardian holds a representation of the neighborhood it controls via predicate guarding(A) that identifies the agents in the neighborhood. Guardians at both ends of a movement (A1) and of a communication (A2)
necessarily check the corresponding requests. A movement request $\text{mobRe}q(A, O, P, S t)$ records the information needed by the guardians involved in the movement process to know who is moving, its origin location, and the requested profile. Similarly, a message request $\text{msgRe}q(M, S, P, S t)$ specifies the message body, the sender, and the profile of the receiver. Variable $S t$ specifies the state of the request. Predicate $\text{in}(M, S)$ represents the receipt of a message $M$ from sender $S$. Predicate $\text{toBeVetoed}$ represents the decision of a guardian to veto a component action, like a movement, or a communication.

The movement of an agent is represented by a change of guardian. Predicate $\text{moveTo}(P, H)$ represents the commitment of an agent to move to a neighborhood satisfying profile $P$. $H$ represents the agent’s history when the request is made. The history is introduced to formalize subjective mobility, as discussed below. The commitment of an agent to move can result in an effective movement or in an exception ($M$). A movement can occur only to a target neighborhood that satisfies the requested profile. Satisfaction is decided by one of the guardians involved in the movement ($M1$). Exceptions follow the decision of a guardian along the route of a movement ($M2$). A movement request is attributed to agent $A$ only if A actually committed to move ($Id1$). To express that mobility is subjective, we need to bind each movement to a corresponding request. This can be done by keeping track of the history of any agent, as a list of the locations it has gone through (predicate $\text{history}(H)$), and by labelling each movement request with the current history ($S1$). We subject a movement to an update of the agent’s movements history ($S2$). Moreover, the history can be updated only if a corresponding movement request exists ($S3$). Initially, the history of an agent is empty ($S4$).

The properties of communication are similar to those of mobility. Predicate $\text{out}(M, P)$ represents the intention of a component to send a message $M$, and $P$ specifies the profile of the receiver. Even in this case the request can result in a successful delivery or in an exception ($C$). Conversely, a message is received only if actually sent and if the receiver satisfies the profile ($C1$). Finally, exceptions occur only because of a decision of one of the guardians involved in the delivery ($C2$). There can be a communication request attributed to a sender $S$ only if $S$ actually committed to communicate ($Id2$).

Predicate $\text{satisfies}$ relates names and profiles. It is used to model the choice of a target neighbourhood of a movement or the receiver of a communication based on the given profile ($M1$ and $C1$). The axioms do not constrain the choice to be unique. This freedom permits to accommodate, as a refinement, unicast, multicast, and broadcast communications. An example refinement is discussed in Section 6.1. On the other hand, in the case of mobility, the axioms for subjectivity and non-uniquity guarantee uniqueness of the target. A profile is specified as a set of constraints, and distinct profiles can have overlapping constraints. If a component satisfies a profile, it means that it satisfies all the constraints that define the profile ($Sat2$). If a set of components satisfies a profile, it means that all the components in that set satisfy that profile ($Sat1$).
<table>
<thead>
<tr>
<th>$A_0$</th>
<th>$U \overline{A}(\text{guardedby}(G_1) \land \text{guardedby}(G_2) \rightarrow G_1 = G_2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S1$</td>
<td>$A(\text{moveTo}(P, H) \rightarrow \text{history}(H))$</td>
</tr>
<tr>
<td>$S2$</td>
<td>$A(\text{guardedby}(G_1) \land \text{history}(H)) \text{UNLESS } A(\text{guardedby}(G_2) \land \text{history}(G_1 \mid H))$</td>
</tr>
<tr>
<td>$S3$</td>
<td>$A(\text{history}(G \mid H)) \text{ BECAUSE } A(\text{moveTo}(P, H))$</td>
</tr>
<tr>
<td>$S4$</td>
<td>INIT $A(\text{history}(t))$</td>
</tr>
<tr>
<td>$S5$</td>
<td>$\overline{A}(\exists G, H, \text{guardedby}(G) \land \text{history}(H))$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$C_0$</th>
<th>$ILA \forall A, \exists G, \text{INIT } A(\text{guardedby}(G)) \land G(\text{guarding}(A))$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M$</td>
<td>$A(\Delta \text{moveTo}(P, H) \land \text{guardedby}(O)) \text{ LEADS } TO}$</td>
</tr>
<tr>
<td></td>
<td>$A(\Delta \text{guardedby}(T) \lor \Delta \text{exc}(\text{mob}(A, O, P, D)))$</td>
</tr>
<tr>
<td>$M1$</td>
<td>$A(\text{guardedby}(T)) \text{ BECAUSE } A(\text{moveTo}(P, H)) \land$</td>
</tr>
<tr>
<td></td>
<td>$G_1(\text{mob} \text{ Re } q(A, O, P, St) \land \text{ satisfy } ({T} P)) \lor G_2(\text{toBeVetoed}(\text{mob}(A, T, P), D))$</td>
</tr>
<tr>
<td>$A1$</td>
<td>$A(\text{guardedby}(T)) \text{ BECAUSE } T(\text{mob} \text{ Re } q(A, O, P, St_1)) \land$</td>
</tr>
<tr>
<td></td>
<td>$O(\text{mob} \text{ Re } q(A, O, P, St_2)) \lor G(\text{toBeVetoed}(\text{mob}(A, T, P), D))$</td>
</tr>
<tr>
<td>$M2$</td>
<td>$A(\text{exc}(\text{mob}(A, O, P, D))) \text{ BECAUSE } A(\text{moveTo}(P, H)) \land$</td>
</tr>
<tr>
<td></td>
<td>$G(\text{mob} \text{ Re } q(A, O, P, St)) \land G(\text{toBeVetoed}(\text{mob}(A, T, P), D))$</td>
</tr>
<tr>
<td>$Id1$</td>
<td>$G(\text{mob} \text{ Re } q(A, O, P, St)) \text{ BECAUSE } A(\text{moveTo}(P, H))$</td>
</tr>
<tr>
<td>$C$</td>
<td>$S(\Delta \text{out}(M, P) \land \text{guardedby}(G)) \text{ LEADS } TO}$</td>
</tr>
<tr>
<td></td>
<td>$S(\text{exc}(\text{msg}(M, S, P), D))$</td>
</tr>
<tr>
<td>$C1$</td>
<td>$R(\text{in}(M, S)) \text{ BECAUSE } S(\Delta \text{out}(M, P)) \land$</td>
</tr>
<tr>
<td></td>
<td>$G(\text{msg} \text{ Re } q(M, S, P, St) \land \text{ satisfy } ({R} P))$</td>
</tr>
<tr>
<td>$A2$</td>
<td>$R(\text{in}(M, S)) \text{ BECAUSE } G_1(\text{msg} \text{ Re } q(M, S, P, St) \land \text{guarding}(R)) \land$</td>
</tr>
<tr>
<td></td>
<td>$G_2(\Delta \text{msg} \text{ Re } q(M, S, P, St_2) \land \text{guarding}(S))$</td>
</tr>
<tr>
<td>$C2$</td>
<td>$S(\text{exc}(\text{msg}(M, S, P), D)) \text{ BECAUSE } S(\Delta \text{out}(M, P)) \land$</td>
</tr>
<tr>
<td></td>
<td>$G(\Delta \text{msg} \text{ Re } q(M, S, P, St) \land G(\text{toBeVetoed}(\text{msg}(M, S, P), D))$</td>
</tr>
<tr>
<td>$Id2$</td>
<td>$G(\Delta \text{msg} \text{ Re } q(M, S, P, St) \land G(\text{toBeVetoed}(\text{msg}(M, S, P), D))$</td>
</tr>
</tbody>
</table>

| $G_0$ | $N \overline{G_1}(\neg \text{guardedby}(G_2))$                                                   |
|       | $Sat1 \overline{G}(G, \text{satisfy}(\text{Set}, P) \rightarrow (C \in \text{ Set } \rightarrow \text{satisfies}(C, P)))$ |
|       | $Sat2 \overline{G}(\text{satisfies}(C, P) \rightarrow (P_i \in P \rightarrow \text{satisfies\_constraint}(C, P_i)))$ |

Table 1. Mobadl model requirements: the formalisation.

To conclude the section, we briefly comment on the use of vetoes. In the WAN setting, we cannot ignore events like failure or unreachability of hosts: we use vetoes to deal with them explicitly. In a uniform manner, we use vetoes also to model mobility and communication failures due to the access policies of the neighbourhoods. As for routing and security policies, we do not fix a priori the reasons why and when vetoes can be raised, this is left to a refinement and delayed to the moment when all the requirements of a particular system have been collected so to make a choice in this sense possible.
4.4.2 The reference model

In this section, we refine the requirements specification to obtain what will be called the Mob_addr reference model, i.e. the level 1 of the refinement process, a reference axiomatisation that will be the starting point of any specification activity based on Mob_addr, like those presented in Section 6.

In particular, we refine axioms in C0, since our interest is to make more explicit the responsibilities of agents and guardians. Moreover, one of the goals of the refinement steps is to reduce the coordination theory, not in size but in the granularity of the properties that it contains: the aim is to leave in the coordination theory as little as needed and to details as much as possible the role that agents and guardians play in the coordination patterns. The behaviour of agents, guardians, and their coordination is specified by theories A1, G1, and C1, respectively as reported in Table 2. These theories refine A0, G0, and C0, accordingly to the refinement strategy discussed in Section 4.3. Theories A1 and G1 include A0 and G0 respectively, while C1 does not include C0.

Axioms for mobility. When an agent A decides to move, it notifies a mobility request mobReq(A, O, P, i) to its guardian O, specifying the profile P the target neighbourhood must satisfy (Cm1). We use variable St in mobReq(A, O, P, St) to model the state of a mobility request. St takes the following values: i, saying that the request is in an initial stage, u, saying that profile P is still unresolved, or dest(T), where T is the name of the target, once selected. The agent’s request can either trigger a movement process (Gm1), in which case the agent is declared to be moving (Gm4), or be immediately rejected if the agent is already moving (Gm5) or for other reasons that are not specified in the reference model (Gm1).

To intercept double movement requests, predicate moving(A) is used: as seen, the agents that asked to leave the neighbourhood and are not back yet are declared to be moving, moreover, we to avoid two movement requests from the same agent to be received at the same time (Gm7). Once an agent enters a neighborhood, the guardian always either guards it or serves a mobility request from the agents (A3, A4), but not both at the same time (A5), i.e. predicates guarding and moving are mutually exclusive. This way, guardians keep track of all the agents that ever entered the neighbourhood: either they are in and guarded, or out, and thus moving.

If the agent’s movement is rejected for a reason not connected to a double request, the guardian reset the state of the agent (Gm6). If the request is accepted, it is routed until it is resolved (i.e. a destination neighbourhood is selected). Guardians are in charge of finding a destination neighbourhood that satisfies the requested profile (Gm2'), and to route the movement request to the destination guardian (Gm1, Gm2) that takes the agents under its control (Gm3). It can happen that no neighbourhood satisfying P exists or that it exists but it is not reachable. Moreover, any of the involved guardians can veto the request because of some policy (Gm1, Gm2, Gm3). In this case, an exception is thrown to the agent that returns under the control of the origin guardian. At the end of the process, the communication that the movement took place or that an exception occurred is sent to the agent (Cm2, Cm3).

65
Since we chose to base our model on subjective mobility, and to avoid fake mobility requests, movement requests must be actually originated by the involved agent ($Cm1'$, $Gm1'$, $Gm2'$, $Gm4'$); for similar reasons vetoes can be communicated to an agent only if some guardian received a movement request from that agent and rejected it ($Cm3'$, $Gm1_2$ $3'$, $Gm5'$). To make consistent the relation between the location awareness of agents and the knowledge guardians have of their agents, an agent can know where it is only after the relative guardian has taken it under its control ($Cm2'$). Apart from the initial state, guardian has the control of an agent only if the agent asked to enter its neighbourhood or it the agent was previously in its neighbourhood and tried unsuccessfully to go away ($Gm3$ $6'$). Axioms $Gm8$ and $Gm9$ relate the variables that model the state of a request and the types of a veto to all their respective possible values.

**Axioms for communication.** When an agent $S$ wants to send a message $M$, it notifies a communication request $msg Req(AM, S, P, u)$ to its guardian $G$, specifying the profile $P$ the receiver must satisfy ($Cm1$). We use variable $St$ in $msg Req(AM, S, P, St)$ to model the state of a communication request. $St$ takes the following values: $u$, saying that profile $P$ is still unresolved, $rec(R)$, where $R$ is the name of the receiver, once selected. The agent’s request either can trigger a communication process or be immediately rejected for reasons that are not specified in the reference model ($Gc1$).

If the request is accepted, it is routed until it is resolved (i.e. a receiver is selected). Guardians are in charge of finding a receiver that satisfies the requested profile ($Gc1_2'$), and to route the communication request to the guardian of the receiver ($Gc1$, $Gc2$) that delivers the message ($Cc2$). It can happen that no receiver satisfying $P$ exists. Moreover, any of the involved guardians can veto the request because of some policy ($Cc1$, $Gc1$). In this case an exception is thrown and communicated to the sender.

To avoid fake communication requests, message delivery requests must be actually originated by the involved agent ($Cc1_1'$, $Cc1_2'$, $Cc2'$); for similar reasons vetoes can be communicated to an agent only if some guardian received a communication request from that agent and rejected it ($Cc3'$, $Gc1_2'$). Axiom $Gc3$ relates the variable that model the state of a request to all its possible values.

**Initial state.** The initial state of agents and guardians is constrained not to contain spurious data like exceptions, communication or movements commitments and message or movement requests ($Ia$, $Ig$).

**Discussion on the axiomatisation.** The reference model does not describe how messages or agents reach their target, or why and where an exception is thrown: the axioms only make clear that messages can be delivered, movements can be realized, and that any of the guardians involved in a communication or a movement, can veto it. In addition, the guardians at both ends are necessarily involved, which other guardians may be involved is fixed by the routing and security policies at lower refinement levels. The formulae introduced there detail the guardians, and take care of what happens in between the sender and target guardians, who raises exceptions, and why. In
particular, the general axioms do not describe how to deliver a message to a moving target. However, they imply that any refinement either guarantees the control stabilization property (i.e., that any message actually reaches its target, if not vetoed) or has ways to stop tracing a target which is moving too fast, and eventually notify a veto (e.g., after a given number of hops between guardians).

We already know that initially every guardian knows the agents in its neighbourhood, and every agent knows the name of its guardian. S5 guarantees that no agent can assert to be “lost”, and Cm2’ guarantees that an agent can assert to be guarded by a guardian T only after T has asserted to guard the agent. Dually, as soon as a guardian controls a new agent, it notifies the agent that thus knows its new guardian (Cm2). The interplay of guarding, moving and guardedby is such that the model does not constrain implementation choices with respect to mobility, e.g., there is no need to stop the execution of an agent immediately after a request to move, and the implementation can wait to do so until permission to leave is granted by the local guardian. The model has nothing to say on the local computations in the meantime, it only entails that communication and mobility requests are dealt with differently: communications are regularly served, while mobility attempts are captured as double requests.

| A1  | 1a INIT A(¬moveTo(P1, H) ∧ ¬out(M, P2) ∧ ¬exec(T, D)) |
| C1  | ILA ∀A∃G.INIT A(guardedby(G)) ∧ G(guarding(A))  |
|     | Cm1 A(ΔmoveTo(P, H) ∧ guardedby(O)) LEADS _TO O(Δmob Re q(A, O, P, i)) |
|     | Cm2 T(∧guarding(A)) LEADS _TO A(∧guardedby(T)) |
|     | Cm3 G(ΔtoBeVetoed(mob(A, O, P), D)) LEADS _TO A(Δexec(mob(A, O, P), D)) |
|     | Cm1’ O(mob Re q(A, O, P, i)) BECAUSE A(ΔmoveTo(P, H) ∧ guardedby(O)) |
|     | Cm2’ A(∧guardedby(T)) BECAUSE T(∧guarding(A)) |
|     | Cm3’ A(Δexec(mob(A, O, P), D)) BECAUSE G(toBeVetoed(mob(A, O, P), D)) |
|     | Cc1 S(Δout(M, P) ∧ guardedby(G)) LEADS _TO G(Δmsg Re q(M, S, P, u)) |
|     | Cc2 G(Δmsg Re q(M, S, P, rec(R) ∧ guarding(R)) LEADS _TO R(Δin(M, S)) ∨ G(ΔtoBeVetoed(msg(M, S, P), D)) |
|     | Cc3 G(ΔtoBeVetoed(msg(M, S, P), D)) LEADS _TO S(Δexec(msg(M, S, P), D)) |
|     | Cc1 _1’ G(Δmsg Re q(M, S, P, u)) BECAUSE G₁(Δmsg Re q(M, S, P, u) ∧ guarding(S)) |
|     | Cc1 _2’ G(Δmsg Re q(M, S, P, u) ∧ guarding(S)) BECAUSE S(Δout(M, P) ∧ guardedby(G)) |
|     | Cc2’ R(Δin(M, S)) BECAUSE G(Δmsg Re q(M, S, P, rec(R) ∧ guarding(R)) |
|     | Cc3’ S(Δexec(msg(M, S, P), D)) BECAUSE G(toBeVetoed(msg(M, S, P), D)) |
| A3  | G(guarding(A)) UNLESS G(moving(A)) |
| A4  | G(moving(A)) UNLESS G(guarding(A)) |
| A5  | G(¬(guarding(A) ∧ moving(A))) |
Another issue that is left unspecified in the general model, is how exceptions are routed: the model only requires that they be delivered. A simple refinement could impose that also exceptions are delivered via the
destination guardian. Finally, the general model is insecure, and malicious agents could fake communication and mobility requests wrapping them in messages. Again, the specification of suitable safety properties to avoid these misbehaviours is left to further refinements.

To prove that the reference model is a correct refinement of the requirements specification, we must check that all the properties expressed by the axioms of theory $C_0$ can be derived as theorems using the axioms of the reference model. The correctness of this refinement has been proved using the proof assistant MaRK, presented in Section 5, and all the proofs can be found in Appendix B.
4.5 A comparison with related models

This section has the aim of clarifying the similarities and the differences between the models presented in Section 2.2 and Mobadv; moreover, it relates the latter even to other models from the literature that did not have the same direct influence on it but that propose similar solutions to cope with common problems. For the sake of a clear presentation, four main topics of discussion are identified even if a sharp distinction among these topics is not feasible, since too many interconnections and influences characterize their relationships. The topics are:

- *architectural choices*, which regard the structure of the model, i.e. how things are organized and following which concepts are factorized;
- *mobility*, which has to do with which kind of mobility is modelled and how;
- *communications*, which gathers considerations about different models of communications and relates them to the specific setting of interest;
- *security*, which deals with how communications and mobility are managed to provide a proper treatment of security related issues.

4.5.1 Architectural choices

In Mobadv a system is distributed on a net of elaboration nodes that are called neighbourhoods. The concept of ambient of Mobile Ambients deeply influenced that of neighbourhood. Both are means to express security and location awareness and to model administrative domains. Both permit to model several crucial issues of mobility, for instance, an administrative domain where computational entities are under the control of a single authority, a naming domain where computational entities adopt a given naming policy, and an address space. Anyway, some differences exist that distinguish the neighbourhood from the ambient. A first departing point is the lack of an a-priori topological structure in the net of neighbourhoods. While ambients are organized in a structure that can dynamically evolve but that always maintains a tree-like shape, the topology of the interconnections between neighbourhoods is a graph. This is a more flexible choice, since the designer can exploit the routing policies of guardians to refine the Mobadv reference model can and describe, for instance, hierarchical models, including nested neighbourhoods. A similar approach has been taken in one of the proposed implementations for the Safe Ambients calculus that is based on flattened version of the ambients hierarchy and on the exploitation of the hierarchical structure with routing rules [Bib 87]. Analogous results can be obtained in Mobile UNITY by the introduction of proper coordination rules in the Interactions section to define the policies ruling the changes of value of location variables. However, the Interactions section of Mobile UNITY heavily relies on the mechanisms of variable sharing and on a *global* knowledge of the states of different entities. In Mobadv, the routing process is
the result of the coordination of several routing policies, each belonging to one of the involved guardians. This choice allows eliminating the need for a knowledge about distributed states that relies on synchronous communications and increases the modularizations of systems description. See also the considerations about communications and security for a better treatment of this aspect. The treatment of the topology of localities in Mob\textsubscript{adtl} is close to that of KLAIM, in particular to that of Open KLAIM where the links between nodes can be modified at run-time and are not characterized by a fixed structure.

Another aspect that differentiates Mob\textsubscript{adtl} from Mobile Ambients and KLAIM is the dynamicity of localities. While ambients can be created and dissolved, and KLAIM nodes can be created and linked to sites, the current notion of neighbourhood is not yet complete, since important requirements like the ability of defining new neighbourhoods are not covered. Similar considerations hold for the impossibility of creating new agents or eliminating existing ones. Some work has been done to this regard in [Bib 86] where the author explores an approach similar to that proposed in [Bib 20] for Mobile UNITY, but further explorations in this direction are needed.

The distinction between agents and guardians and the delegation to the latter of all the issues regarding security, routing and mobility follows the direction of a clean distinction between computation and coordination. The same is fostered in KLAIM by the two-level view of systems design that is split between the programmer in charge of defining the functional aspects and the coordinator that manages security and net-infrastructure.

4.5.2 Mobility

In Mob\textsubscript{adtl}, agents move autonomously in and out of neighbourhoods, depending on the security policies of the involved guardians, and always maintain the knowledge of their current guardian. This combination of subjective mobility and location awareness is close to that of Mobile UNITY where processes are augmented with a special variable whose value models the process current position, and can autonomously change the value of the location variable thus changing their location. Since sharing the location variable would result in troubles with regard to the mechanisms of sharing itself, which is based on co-location, even in this model mobility cannot be but subjective. On the other hand Mobile UNITY does not take into account security concerns and in this reveals it origin as a formalism to describe the movement of mobile hosts in a physical space where no administrative domain are present.

KLAIM constructs allow realizing both subjective and objective mobility: a process can move itself to a new location or it can spawn a new process at the desired location. Processes, in any case, move in isolation not carrying with them other active processes.

In Mobile Ambients, like in KLAIM, Mobility can be both subjective and objective but in this case, processes move with the ambient they live in and this means that the whole ambients hierarchy originating from the moving ambient is moved along, and so all the active processes contained therein. Once again, the difference between Mob\textsubscript{adtl} and Mobile Ambients relates somehow to the topology of the localities. Indeed, even in this case, the model described in Mobile Ambients could be implemented in Mob\textsubscript{adtl} by defining a coordination process between an agent and its guardian leading to a change
in the interconnections between the agent’s current neighbourhood and the others. This would introduce in Mob_{adlt} an objective mobility like the one in Mobile Ambients making neighbourhoods move with their content of agents. A similar approach can be exploited to model physical mobility. Indeed changes in the links between guardians could occur independently from the will of agents and would represent the movement of the physical devices on which the neighbourhoods are implemented.

Mobility is seen also as a change in communication links. Formalisms like the $\pi$-calculus do not model localities explicitly but define movements of mobile entities in terms of a change in the available connections to other entities, hence, exchanging channels names is the mean that allows implementing mobility. This is close to physical mobility but seems to be too restrictive in a setting like that of logical mobility where a location is much more than the collection of the available channels and comprises things like a context of data and computational resources, channels and security constraints.

**4.5.3 Communications**

The communication models of Mobile UNITY and Mobile Ambient both rely on local communications. In the former processes can communicate only when co-located, in the latter processes can output messages only in the current ambient. Even if they differ in the fact that communication in Mobile UNITY are synchronous and in Mobile Ambients are asynchronous, both these models deny the possibility of having remote communications if not by coding them explicitly using messenger processes/ambients that receive a message to deliver and a route to follow to reach the intended receiver. This choice seem to be too reductive and, with regard to Mobile UNITY, the synchronicity of communications, which are based on variable sharing, is definitely not the best solution in mobile settings be them physical or logical.

In Mob_{adlt}, communications are based on asynchronous message passing that permits to keep the model abstract from any specific communication protocol. Synchronicity can be coded through explicit handshakes and locality can be forced through proper security policies enacted by guardians that block any message delivery request for receiver that are not in their neighbourhoods.

The mechanism of profile resolution used in Mob_{adlt} communications and mobility, resembles the use of logical localities in KLAIM as a mean to abstract net resources from their physical addresses, but it goes beyond this allowing to identify an entity, possibly even a neighbourhood, by specifying a set of properties that this entity must satisfy.

Channel-based communications from models like that proposed by the $\pi$-calculus and all its derivatives are not directly taken into account in Mob_{adlt}. Channels as means of direct and free communications between entities can still be included in Mob_{adlt} as a proper coordination between the guardians involved in the delivery of messages between two entities. These guardians can be instructed, or requested, to let all messages pass without applying any security policy. Moreover, the guardians can store routing data to preserve the channel regardless possible movements of the sender or the receiver. A refinement with these properties has been presented in [Bib 87], but strong doubts remain about the opportunity of defining a mechanisms as rigid as
that of channels in a highly mobile setting where security plays a relevant role and where different security and routing policies coexist.

Linda-like models base their success on anonymous asynchronous communications and stress the temporal and spatial decoupling of senders and receivers. While the temporal decoupling deriving from asynchrony is an appealing feature and is one of the characteristics of Mob$_{adlt}$ as well, a deep interest in security led to explicitly identify the sender and, at a certain extent, the receiver of a message. Obviously, naming in messages exchange can be modelled also in Linda-like formalisms, but we preferred to have it as a base to our communication model. Anyway, a form of spatial decoupling is guaranteed by the fact that communications can be remote and senders do not have to know anything about the locality of the receivers; moreover, in the reference model senders can move after sending a message regardless the actual delivery of the message.

4.5.4 Security

A Mob$_{adlt}$ guardian acts as an interposition interface among agents and neighbourhoods: it specifies and implements communication and movement policies. In other words, guardians monitor the agents and limit the resources they can use. An example of how security policies can be specified in Mob$_{adlt}$ can be found in [Bib 88]. A need for a considerable flexibility in allowing the coexistence of even deeply different security policies come naturally in setting like that of WANs. With regard to this, an important aspect of Mob$_{adlt}$ is that it provides each guardian with the ability to define its own policy. The approach has several advantages. First, it supports interoperability: different computing environments may need different security requirements. Moreover, nothing prevent from enforcing the security policies of the guardians by using different mechanisms. On the contrary, most models for secure mobile systems base their policy on a specific mechanism. These can be:

- **Secure type programming languages.** Capability based type systems have been recently introduced to specify and enforce access control policies of mobile agents [Bib 8, Bib 14, Bib 34]. The specification of the security policy can be transformed into a secure typed interface and implemented in a secure type language: the soundness of the type system ensures that program does not violate the security requirements implied by the type interface.

- **Cryptography.** The security policy has the role of establishing trust among guardians. Standard implementations exploit cryptography to enforce authority and identity of components. Cryptographic mechanisms are included in modern programming languages, e.g. the Java APIs [Bib 89, Bib 90, Bib 91, Bib 92, Bib 93].

- **Proof-Carrying Code.** Necula [Bib 94] has introduced an approach to ensure correctness of mobile code with respect to a fixed safety policy: code producers provide the code with a proof of correctness that code consumers check before allowing the code to execute. The security policy can be transformed into a safety policy and a Proof-Carrying Code mechanism is used to determine whether the
component supplied by a different guardian satisfies the safety policy.

Notice also that Mobadlt works even if the guardians do not trust each other. Each neighbourhood can enact its own policy and when inter-neighbourhood interactions occur, agents must cope with the requirements of a dynamically defined global security policy that is the result of the coordination of the policies of all the guardians involved. This accommodates well a characteristic of WAN applications: any security related decision could not depend on some knowledge of the entire current state of a WAN application; a more realistic approach, instead, identifies the portion of the state of the WAN application that is potentially relevant with regard to security policy decisions.

In Mobile Ambients, the possibilities of interacting with an ambient are hard-wired coded in the specification calculus and defined by the name-based capability mechanism. Any particular security policy that does not have a direct counterpart in this mechanism must be explicitly programmed, sometimes leading to clumsy and hard to read and understand pieces of code that interfere with the functional part of the description of a process. The neighbourhood, on the other hand, is just a locality identified by the name of its guardian, and all the interactions that may occur with it are ruled not by a fixed mechanisms but by the security policy that the guardian is called to enact. Even in this case the designer must explicitly describe the desired security policy, but the declarative character of DSTL(x) specifications makes it easier to express and read the needed properties and no mechanism if fixed a priori as the one to be used to describe policies. Moreover, a clear separation between coordination, security in this case, and computation is realized, thus fostering the factorizing of specifications for reuse, and separating the interventions of different expertises in the design of a system.

As far as Mobile UNITY is regarded, the same considerations made about architectural choices, communications and mobility hold in this case, since security in Mobile UNITY is not directly taken into account and any introduction of a security policy would pass through the Interactions section.

The approach to security of Mobadlt is close to, even if not directly influenced by, the work on Execution Monitoring [Bib 95] as a mean to enforce security. Execution Monitoring is an enforcement mechanisms that works by monitoring execution steps of a target system terminating its execution if it is about to violate the security policy being enforced. This mechanism includes things like security kernels, reference monitors, firewalls, and most operating system and hardware-based enforcement mechanisms. The targets may be objects, modules, processes, subsystems, or entire systems; the execution steps monitored may range from fine-grained actions (such as memory accesses) to higher-level operations (such as method calls) to operations that change the security-configuration and thus restrict subsequent execution. This approach relies on the parallel execution of the target and of a security automata: each action performed by the target produces a symbol that is assed to the automata, if the automata can make a transition with that symbol the target is allowed carrying on it execution, otherwise it is terminated. Mobadlt guardians are a conceptual tool to identify security related features, more than an enforcement mechanism for these features. So, while Execution Monitoring prescribes well defined rules to
enact security policies and clearly characterizes which policies can be enacted by those rules, in Mob\textsubscript{adtl} it is simply said that all the action of agents must pass some security related check, that security policies are location-dependent and that it must be possible to make different security policies coordinate. No commitments to particular implementation choices are made. Moreover, in the case of security violations, the reference model provides the designer with the chance of letting the agents have some information about the reasons that made a guardian refuse their actions.

In [Bib 96] the definition of Guarded Boxed Ambients is based on an approach that is related to the work on Execution Monitoring and that has been partially inspired by the guardians of Mob\textsubscript{adtl}. The authors give an extension of ambient-based calculi based on the introduction of one guardian for each ambient. The role of guardians is to monitors the activity of processes and sub-ambients and the interactions that occur with the external environment, possibly preventing the execution of action when they do not adhere to the prescribed security requirements. Guardians in this work at like security automata in Execution Monitoring but can and must coordinate in order to allow ambient interactions; with regard to this, the introduction of guardians leads to a security treatment model similar to that of Mob\textsubscript{adtl}, where local security policies dynamically combine to define global security policies.
5 Verification of Mob_adtl systems

As pointed out in Section 3, a successful introduction of formal methods in the common practices of software engineering passes necessarily through the definition and development of support tools. Consequently, we developed MaRK (Mob_adtl Reasoning Kit) a proof assistant for ADSTL(x) specifically tailored for the Mob_adtl model. It has been developed using Isabelle [Bib 98, Bib 99] a generic theorem prover that has been instantiated to support the reasoning in several object-logics, including higher order logic and modal logics. A previous version of MaRK based on the logic Oikos-adtl has been presented in [Bib 97]. The following sections present a quick Isabelle tutorial and the current implementation of MaRK. The structure of MaRK and the way we coded in Isabelle DSL(x), ADSTL(x) and the Mob_adtl model are explained in detail. Moreover, the process of verification using Mark is illustrated through a description of some proofs from of theorems of the logics and of the model properties. The complete code of the theories presented in this section and a the complete collection of all the proofs performed in MaRK can be found in Appendix A and Appendix B.

5.1 The theorem prover Isabelle

Isabelle is a generic theorem prover. It has been instantiated to support reasoning in several object-logics among which First Order Logic on which the work on the verification of Mob_adtl is based. The first distribution of Isabelle dates back to 1986; the version we refer to is Isabelle99, which is not the most up to date since this tool is still under development and continues to change and improve.

To define a new logic in Isabelle, one must specify declaratively its syntax and inference rules. Isabelle provides control structures for expressing search procedures; moreover, it integrates several generic tools, like simplifiers and classical theorem provers, which can be applied to the object-logics.

The next two sections explain how to define a logic and how to build proofs. These sections are meant to give a quick tutorial on Isabelle so to ease the comprehension of the work presented in Section 5.2. Anyway, this tutorial does not have any commitment to completeness and readers are invited to refer to Isabelle documentation that can be found at Bib 98.

5.1.1 Defining a logic

Using Isabelle to define a logic, or better, extending the generic theorem prover Isabelle to reason on a particular logic, means to give syntax to write the formulae of the logic, and to give rules to be applied when trying to perform proofs in this logic. This task is accomplished by creating a theory collecting all these information in a file whose name is the theory's followed by a .thy. Table 3 shows the generic structure of a Isabelle theory.
First a theory \( T \) is declared as an extension of existing theories \( S_1 \ldots S_n \). The details of the new logic are given by introducing new classes, new types with their arities, constants and rules. Concrete syntax for constants can be defined while the translations section specifies rewrite rules on abstract syntax trees to handle notations and abbreviations. After the theory declaration, concluded with the keyword end, there is the ML section that may contain ML code to perform arbitrary syntactic transformations. All the sections presented in Table 3 can be omitted, repeated and presented in any order.

The fact that a logic is defined an extension of other logics means that it inherits all the types, constants, syntax etc. defined in the logics it extends. The most basic Isabelle theory is Pure that contains all the elements of Isabelle meta-logic, the basis of Isabelle proving facilities.

Once the file of a theory \( T \) has been written, the user can tell Isabelle to process it using the command use_thy "T". When reading the \( T.thy \) file, Isabelle writes a corresponding ML file \( T.thy.ML \), reads it and, if no errors occurred, deletes it. This declares the ML structure \( T \), which contains a component \( thy \) denoting the new theory, a component for each rule, and everything declared in the ML section of the theory. After building this structure, Isabelle looks for and reads, if present, the file \( T.ML \) which contains all the proofs performed using theory \( T \) built as explained in the next section. All the proofs are performed and the results stored to be used again as theorems.

### 5.1.1.1 Rule declarations

As an example of how to define a structure like the one in Table 3 let’s examine the extract from the code of theory PC reported in Table 4. Theory PC extends theory FOL, which is the Isabelle implementation of First Order Logic, by providing some interesting predicate calculus theorems: contraposition, introduction and elimination of double implication, and de morgan. The rule

\[
\text{ContrapositionPN } "(F1\rightarrow F2)\rightarrow\neg(F2\rightarrow\neg F1)"
\]

where operators \( \rightarrow \) and \( \neg \) stand for \( \rightarrow \) and \( \neg \) respectively, is an example of Isabelle rule with one premise and follows the pattern
rule_name "rule_premise==>rule_consequence"

that corresponds to the form \[
\frac{\text{rule_premise}}{\text{rule_consequence}}
\]

As an example of the shape of Isabelle rules with more than one premise, consider

\[
\text{IffI } "[|F1-->F2;F2-->F1|]-->F1<->F2"
\]

where the operator <-> corresponds to \(\leftrightarrow\). The declaration of this rule follows the pattern

rule_name "[|rule_premses|]==>rules_consequence"

that corresponds to the form \[
\frac{\text{rule_premise}_1,...,\text{rule_premise}_n}{\text{rule_consequence}}
\]

Rule IffI is called an introduction rule, since is introduces in its consequence and operator that does not occur in the premises. The corresponding elimination rules, which remove an operator from the premises substituting it in the consequence with other operators, are

\[
\text{IffE_right } "F1<->F2==>\(F1-->F2\)"
\]
\[
\text{IffE_left } "F1<->F2==>\(F2-->F1\)"
\]

that derive an implication from a double implication. The following rule is a Isabelle version of one of the de morgan’s rules

\[
\text{de_morgan1 } "¬(A \& B) == ¬A | ¬B"
\]

In the code of de_morgan1, operators & e | stand respectively for \(\land\) and \(\lor\), defines a meta-logical equivalence between \(¬A|¬B\) and \(¬(A \& B)\) that can be used to rewrite one in the other during proofs.

```
PC = FOL +

rules
... ContrapositionPN "(F1-->F2)==>¬(F2-->¬F1)"
... IffI "[|F1-->F2;F2-->F1|]-->F1<->F2"
IffE_right "F1<->F2==>\(F1-->F2)"
IffE_left "F1<->F2==>\(F2-->F1)"
... de_morgan1 "¬(A \& B) == ¬A | ¬B"
end
```

Table 4. Rules declarations from theory PC.

78
5.1.1.2 Type and constant declarations

As an example of how to define new types and constants let’s examine the extract from the code of theory DSL reported in Table 5.

```
DSL = PC +
types
  component_name
arities
  component_name ::= term
consts
  l ::= [component_name,o] => o
  co_l ::= [component_name,o] => o
defs
  co_l_def "co_l(M,F) == ~l(M,~F)"
  l_def "l(M,F) == ~co_l(M,~F)"
rules
(* DSL axioms *)
  K "co_l(M,F1-->F2)-->(co_l(M,F1)--->co_l(M,F2))"
end
```

Table 5. Type declarations from theory DSL.

This theory extends theory PC adding the new type `component_name`, two constants with their definitions, and some rules of which only rule K is reported in the table. The declaration of type `component_name` corresponds to the most basic kind of type declaration: a type constructor is defined that does not have any argument. Consequently, the arity declaration for this type is just the class `term` to which this type is defined to belong.

Constants `l` and `co_l` are defined are binary operators that get an element of type `component_name` and an element of type `o` as arguments and give back an element of type `o` as result. Type `o` is the Isabelle type for logic formulae, i.e. for everything that can be said to be true or false and is allowed to be used as a premise or a consequence of a rule.

The `defs` sections of this theory contains the mutual definition of the two constants defined in the const section. These definitions can be used during proofs to rewrite terms built using `l` and `co_l`.

Rule K, is an example of the syntax of terms written using `co_l`. Constants from theories built extending FOL always have the syntax

```
const_name(argument_1,...,argument_n)
```

The code reported in Table 5 is just a simple example of type declarations that in their most general form shape like

```
Types  tid_i id_i
...
  tid_n id_n
```
where \( id_1, \ldots, id_n \) are the type identifiers, like \( \text{component}\_\text{name} \), and \( tid_1, \ldots, tid_n \) are type arguments lists, which do not occur in the declaration of type \( \text{component}\_\text{name} \). Each \( id_i \) is declared as a type constructor with arguments \( tid_i \). Following this pattern, the declaration of a list type could look like

```
types 'a list
```

where \( 'a \) is a variable type and means that the constructor \( \text{list} \) can be used to build lists of element of any type. The declaration of a tree type could be

```
types ('a, 'b) tree
```

meaning that the constructor \( \text{tree} \) receives two element of different type.

The declaration of type arities, which take its simplest form for type \( \text{component}\_\text{name} \), follows in general the pattern

```
antages tycon_i :: arity_i
...
   tycon_n :: arity_n
```

The arity declaration for types \( \text{list} \) and \( \text{tree} \) would be the following

```
antages list :: (term)term
tree :: (term,term)term
```

meaning that the elements of these types are of class \( \text{term} \) provided that the arguments of the constructors are of that class.

Once types, together with their arities, are defined, constants can be declared to build elements of those types. As an example consider

```
1 :: [component_name,0] => o
```

that is the the declaration of the constructor used to represent the DSL(\( x \) locality operator. The meaning of this declaration is that \( 1 \) is a binary constant that gets an element of type \( \text{component}\_\text{name} \) and an element of type \( o \) to build en element of type \( o \). The general pattern for a constant declaration looks like the following

```
consts c_i :: \( \tau_i \n...
   c_n :: \( \tau_n \n```

where \( c_i, c_n \) are constants and \( \tau_i, \tau_n \) are types. Constants can be given a concrete infix or mixfix syntax like in the following examples
• consts and :: [o,o] => o (infixl "&" 35)
  the binary and constant is defined, and it is provided with a
  concrete syntax that defines the infix operator & that is left
  associative and has priority 35; using and we can write formulae
  like and(A,B), while using its concrete syntax we can write A & B;
• consts If :: [o,o,o] => o ("if _ then _ else ")
  the constant If is declared, and it is provided with a concrete
  syntax in which each _ is used as a placeholder for an argument;
  using If we can write If(P,Q,R), while using its concrete syntax
  we can write if P then Q else R;
•consts leads_to :: [o,o] => o (" _ LEADSTO ")
  from theory dDSTL; the binary constant leads_to is defined, and
  it is provided with a concrete mixfix syntax; using leads_to we
  can write formulae like leads_to(F1,F2), while using its
  concrete syntax we can write F1 LEADSTO F2;

Once a constant is given a concrete syntax, the concrete syntax can be used
both to define rules and to state goals to be proved in the proof sections. Even
the output provided by Isabelle during proof sections will be based on the
concrete syntax.

5.1.2 Performing proofs

Proving in Isabelle can proceed forward or backward. Proving forward
means to build new theorems using the already defined rules and operators
RS and RSN to join them. The basic idea is that a rule is used to resolve one
of the premises of another rule thus to obtain a new rule like in

    conjunct1 RS mp;

where conjunct1 "P&Q==>P" corresponds to the conjunction elimination
rule

\[
\frac{P \land Q}{P}
\]

and mp "[[P-->Q; P]]==>Q" is the Isabelle version of modus ponens and
corresponds to the rule

\[
\frac{P \rightarrow Q \quad P}{Q}
\]

The result of this construction is the theorem

    [|(P-->Q) & Q1; P|] ==> Q

that corresponds to
\[
(P \rightarrow Q) \land Q, \quad P \\
\hline
\]

Proving backward means to state a goal, i.e. the theorem to prove, and to reduce it to subgoals by applying proper rules. Goals are stated using the command

```
Goal  "formula";
```

that adds formula to the proof state, i.e. the list of subgoals to be proved, as subgoal number 1. The proof proceeds by using tactics to solve the goal. The main tactics for backward proving are:

- **assume_tac** \(i\); it is the tactic that attempts to solve subgoal \(i\) by assumption, i.e. by unifying the consequence of the subgoal with the first of the premises of the subgoal;
- **resolve_tac** \(thms\) \(i\); it is the basic resolution tactic, used for most of proof steps. The \(thms\) represent rules that are resolved against subgoal \(i\). For each rule, resolution forms next states in the proof process by unifying the conclusion with the subgoal and inserting instantiated premises in its place;
- **eresolve_tac** \(thms\) \(i\); it is the tactic that performs elimination resolution, i.e. it acts like resolve_tac but additionally solves by assumption the first premise of the rules it applies, and deletes that assumption from any goal arising from the resolution;
- **dresolve_tac** \(thms\) \(i\); it is the tactic that performs destruction elimination which works forward from a subgoal’s assumptions.

Tactics are applied to a proof state, i.e. the list of the subgoals to be proved, by the command

```
by  tactic;
```

that applies tactic to the current proof state raising an exception if the tactic fails. As an example consider the following proof of \(P \lor P \rightarrow P\). First we state the goal

```
Goal  "P\mid P\rightarrow P";
```

And get the following proof state with one subgoal that corresponds to the main goal

```
1.  P\mid P\rightarrow P
```

Then we apply resolve tac with the rule \(\text{impI} \quad "(P\Rightarrow Q)\Rightarrow P\rightarrow Q"\) to subgoal 1

```
by  (resolve_tac  [impI]  1);
```
and get

1. \( P|P =\Rightarrow P \)

that shows a subgoal with assumption \( P|P \). The next step is to apply \( \text{disjE} \) "\(|P|Q;P=\Rightarrow R;Q=\Rightarrow R|\Rightarrow\Rightarrow R|\)" by issuing the command

\[
\text{(resolve_tac [impI] 1)};
\]

The result of this command is to substitute the previous subgoal with the following three new subgoals that correspond to the instantiation of the premises of \( \text{impI} \)

1. \( P|P =\Rightarrow P1|Q1 \)
2. \( |[P|P;P1|] =\Rightarrow P \)
3. \( |[P|P;Q1|] =\Rightarrow P \)

Each one of these three subgoals is provable by assumption, i.e. by unifying each one of them with one of their assumptions. Let’s prove first subgoal 3 by

\[
\text{(assume_tac 3)}
\]

which unify \( Q1 \) with \( P \), remove subgoal 3, which is proved, and updates the other two subgoals giving as result the following proof state

1. \( P|P =\Rightarrow P1|P \)
2. \( |[P|P;P1|] =\Rightarrow P \)

We apply now the same tactic to subgoal 2 to unify \( P1 \) with \( P \), remove subgoal 2, which is proved, and get the following subgoal

1. \( P|P =\Rightarrow P|P \)

that can be proved by another call to \( \text{assume_tac} \). Once the goal has been proved we can store it by giving it a name with the command

\[
\text{qed "my_theorem"}
\]

that allows us to use the theorem in future proofs referring it as \( \text{my_theorem} \).

Tactics can be combined to build more powerful and structured tactics using the operators, called tactical, provided by Isabelle. The following are the basic tactical:

- \( \text{tac1 THEN tac2; tactical THEN is used to sequentially combine tactics. Applied to a proof state, tac1 THEN tac2 returns the proof state reachable by an application of tac1 followed by an application of tac2;} \)
• tac1 ORELSE tac2; tactical ORELSE is used to express a choice tactic. Applied to a proof state, tac1 ORELSE tac2 return the result of the application of tac1, in not empty; otherwise returns the result of the application of tac2;

• REPEAT tac; tactical REPEAT is used to iterate the application of a tactic. Applied to a proof state REPEAT tac returns the proof state reachable applying tac as long as possible.

Tactical can be used in combination like in

    REPEAT (tac1 ORELSE tac2)

that defines a tactic that repeatedly applies tac1 and tac2, giving priority to tac1.
5.2 MaRK: Mob\textsubscript{adlt} Reasoning Kit

MaRK embodies the methodology to support its intended users, namely system designers, in the verification that their system decomposition, in terms of agents and guardians, is correct with respect to the global system requirements. The tool manages three different kinds of information: theories, tactics, and proofs: theories are the Isabelle translation of Mob\textsubscript{adlt} reference model axioms; tactics are proof stereotypes which can be reused in the verification activity; the proofs includes theorems derived within the assistant, to be used as lemmas. MaRK works as follows. The user supplies it with the system specification. Then, to prove a property, the user proceeds interactively selecting tactics or theorems among those stored by MaRK or by applying logic rules to decompose the current property in sub-properties. When the proof terminates, the user can add it to the proof library. If a new tactic has been singled out during the proof, it can be elaborated and inserted in the tactic library. The advantages of using MaRK include: the ability to keep track of the assumptions a property relies on; the ability to handle standard bureaucratic activities (first order logic reasoning and standard verification techniques); the ability to support the proving pattern that emerges when composing subsystems; the ability to control the overall architecture of applications including refinement steps.

Building MaRK meant to code in Isabelle the logic used to describe the Mob\textsubscript{adlt} model and to organize the Isabelle theories so to reflect the structure at the basis of the refinement process. Following Isabelle rules, the process led to the creation of a fairly complex hierarchy of theories that, for the sake of comprehension, will be split in the next sections into separate sub-hierarchies, one presenting the theories that implement the Isabelle coding of DSL(x) and \Delta DSL(x), the other presenting the theories that represent the Isabelle coding of Mob\textsubscript{adlt} requirements specification and reference model. Theories are presented together with the respective tactics and proofs. In particular, the next sections show how we used MaRK to prove the theorems of DSL(x), those of \Delta DSL(x), and the Mob\textsubscript{adlt} reference model properties.

5.2.1 Coding the logics

The hierarchy of the theories that globally represent the extension of the Isabelle generic theorem prover to \Delta DSL(x) is presented in Figure 3 where each box represents an Isabelle theory (the ensemble of the .thy and .ML files) and the arrows connecting different boxes stand for the extension/inclusion relation existing between the theories. This means that the box labelled FOL stands for the most basic theory and the one that is labelled dDSL represents the theory that comes out as the final result of the extension process.

![Figure 3. Logic theories hierarchy.](image-url)
5.2.1.1 DSL: coding DSL(x)

Figure 3 tells us that theory DSL has been built by extending the FOL theory, i.e. the pre-defined Isabelle theory that implements First Order Logic. Theory PC is a support theory that provides the proofs of several predicate calculus theorems that are exploited in the others theories. DSL is built on FOL (through PC) adding all the elements that contribute to the definition of DSL(x): a type for the components, operators for the modalities, and rules to formalize the DSL(x) axioms. Table 6 reports an extract from the DSL.thy file, the complete code can be found in Appendix A.

The meaning of the code in Table 6 is that DSL extends PC and defines the new type component_name and the two constructors l and co_l that are used to model the localities m and co-localities m of DSL(x) respectively. The type o is used in Isabelle to refer all the logic formulae, i.e. everything that one can say to be true or false. Operators l and co_l take a component name and a logic formula describing a property of a state of the component as arguments. In the defs section of this Isabelle theory, the mutual definitions of l and co_l are given to be used as rewriting rules during the proofs. Unary operator ~ stands for ¬.

```
DSL = PC +
types
  component_name
_consts
  l :: [component_name,o] => o
  co_l :: [component_name,o] => o
defs
  co_l def "co_l(M,F) == ~l(M,~F)"
  l_def "l(M,F) == ~co_l(M,~F)"
rules
(* DSL axioms *)
K "co_l(M,F1-->F2)-->co_l(M,F1)-->co_l(M,F2)"
DSL1 "co_l(M,F<->co_l(M,F))"
DSL2 "~{M=N}-->co_l(M,co_l(N,False))"
Nec "F-->co_l(M,F)"
End
```

Table 6. Theory DSL.

The formalization of the DSL(x) axioms follows. The syntax is fairly intuitive, for example the coding of \( K : \overline{m}(F \to F') \to (\overline{m}F \to \overline{m}F') \) takes the form

\[
K "co_l(M,F1-->F2)-->co_l(M,F1)-->co_l(M,F2)"
\]

that follows the pattern rule_name “rule_code” of Isabelle rules where the symbol --> stands for the logical implication →.

The only remark regarding the Isabelle coding of DSL(x) is about axiom DSL2 that, in its Isabelle version, exhibits an hypothesis stating the fact that
$M$ and $N$ are two distinct components. Actually, this hypothesis is present also in the original version of the axioms where two different letters are used to identify the two components with the assumption, now made explicit, that different letters identify different components. Such variations recur often in the Isabelle coding of both DSL(x) and ΔDSL(x).

### 5.2.1.2 DSL tactics and proofs

Using MaRK, we proved all the DSL(x) theorems cited in Section 4.2.1. Here we explain some of them to give an insight on DSL(x) proof strategies and to show how to use MaRK to reason on properties written in this logic. For the complete code of the proofs and tactics of theory DSL see Appendix B.

The first theorems we proved are those that allow weakening of localized formulae: l_ref and co_l_ref. The former has the form

$$F_1 \rightarrow F_2$$

and is proved as follows in DSL(x)

$$\begin{align*}
K & \quad \overline{m}(\neg F_2 \rightarrow \neg F_1) \rightarrow (\overline{m} \neg F_2 \rightarrow \overline{m} \neg F_1) \\
mp & \quad \overline{m}(\neg F_2 \rightarrow \neg F_1) \rightarrow \overline{m} \neg F_1 \\
\frac{F_1 \rightarrow F_2}{\overline{m} F_1 \rightarrow \overline{m} F_2} \text{ contr.} & \quad \frac{\neg F_2 \rightarrow \neg F_1}{\overline{m}(\neg F_2 \rightarrow \neg F_1)} \text{ Nec} \\
\frac{\overline{m} \neg F_2 \rightarrow \overline{m} F_1 \rightarrow \overline{m} \neg F_1}{\overline{m} F_1 \rightarrow \overline{m} F_2} \text{ def.} \overline{m}, \text{contr.}
\end{align*}$$

In MaRK the proof is the following

```
Goal "F1==>F2==>l(M,F1)-->l(M,F2)";
  by (l2co_l_tac(1));
  by (K_tac_simple(1));
  by (resolve_tac [Nec] 1);
  by (eresolve_tac [ContrapositionPN] 1);
qed "l_ref";
```

First, the theorem is stated as a goal to be proved, and then the steps of the proof are performed and, to conclude, the proved theorem is registered under the name l_ref. The first step to be performed is to rewrite the goal exploiting the definition of the co-locality operator; tactic l2co_l_tac has been defined to achieve this result. In this case, the tactic rewrites the consequence of the goal applying the definition of co_l and contraposition substituting the goal with

$$F1==>F2==> co_l(M,~F2)---> co_l(M,~F1)$$

Then K_tac_simple is called to reduce the goal to

$$F1==>F2==>co_l(M,~F2--->~F1)$$
The following is the code of \texttt{K\_tac\_simple(n)}. The tactic is defined as a function that applies to the $n$th subgoal first \emph{modus ponens}, or better its Isabelle version mp "[P--->Q; P] ==> Q", and then axiom K, as shown in the following code from DSL

\begin{verbatim}
fun K_tac_simple(n) = (
  (resolve_tac [mp] n) THEN
  (resolve_tac [K] n)
);
\end{verbatim}

The last two steps of this proof correspond to the application of \texttt{Nec} that produces the subgoal

\begin{verbatim}
F1-->F2===>~F2===>~F1
\end{verbatim}

and of \texttt{ContrapositionNP "(~F1-->~F2)==>(F2-->F1)"} defined in \texttt{PC} that reduces the consequences of the subgoal to the same form of the assumption \texttt{F1-->F2}. This permits to resolve the former using the latter, and to conclude the proof.

Theorem \texttt{co\_l\_ref} is proved in a similar way by applying tactic \texttt{K\_tac\_simple} and \texttt{Nec}. The following is the pattern of the proof

\begin{align*}
\frac{K}{mp} & \\
\frac{\overline{m}(F_1 \rightarrow F_2) \rightarrow (\overline{m}F_1 \rightarrow \overline{m}F_2)}{\overline{m}F_1 \rightarrow \overline{m}F_2} & \frac{F_1 \rightarrow F_2}{\overline{m}(F_1 \rightarrow F_2)}
\end{align*}

and this is the MaRK version

\begin{verbatim}
Goal "F2-->F1===>co_l(M,F2)==>co_l(M,F1)";
by (K\_tac\_simple(1));
by (eresolve_tac [Nec] 1);
qed "co\_l\_ref";
\end{verbatim}

Once the goal has been stated, the application of K leaves us with

\begin{verbatim}
F2-->F1===>co_l(M,F2==>F1)
\end{verbatim}

that can be reduced using \texttt{Nec} to

\begin{verbatim}
F2-->F1===>F2===>F1
\end{verbatim}

The resolution of the consequence against the assumption concludes this proof.

With these two theorems proved, we can use them during the proofs, or better, we can define tactics to ease their exploitation. Tactic \texttt{co\_l\_ref\_tac} tries to resolve a goal using \texttt{co\_l\_ref}; if this does not give a positive result,
the tactic goes on trying to resolve the goal applying first *modus ponens* and then \texttt{co\_l\_ref}.\footnote{\texttt{co\_l\_ref}\_tac(n) = \}

\begin{verbatim}
fun co_l_ref_tac(n) = (  
    ((resolve_tac [co_l_ref] n))  
  ORELSE  
    ((resolve_tac [mp] n) THEN  
      (resolve_tac [co_l_ref] n))  
); 
\end{verbatim}

A similar tactic, \texttt{l\_ref\_tac}, has been defined for \texttt{l\_ref}. The following MaRK code proves a stronger version of *axiom4* $\overline{F} \overline{\overline{M}} \overline{F} \rightarrow \overline{\overline{M}} \overline{F}$, in which the double implication holds, and uses \texttt{co\_l\_ref\_tac}.

\begin{verbatim}
Goal "co\_l(M,F)\<-->co\_l(M,co\_l(M,F))";
by (resolve_tac [IffI] 1);
  (* --> *)
    by (K_tac_simple(1));
    by (co_l_ref_tac(1));
    by (resolve_tac DSL_AXIOMS 2);
    by (resolve_tac [impI] 1);
    by (IffE_tac(DSL_AXIOMS,1));
  (* <-- *)
    by (K_tac_simple(1));
    by (co_l_ref_tac(1));
    by (resolve_tac DSL_AXIOMS 2);
    by (resolve_tac [impI] 1);
    by (IffE_tac(DSL_AXIOMS,1));
qed "axiom4";
\end{verbatim}

After stating the goal, two subgoals are obtained splitting the double implication by using theorem

\begin{verbatim}
IffI "[| F1<->F2; F2<->F1 |] ==> F1<->F2"
\end{verbatim}

defined in \texttt{PC}. In this way we obtain two subgoals corresponding to the two directions of the double implication

\begin{enumerate}
  \item \texttt{co\_l(M,F)<-->co\_l(M,co\_l(M,F))}
  \item \texttt{co\_l(M,co\_l(M,F))<-->co\_l(M,F)}
\end{enumerate}

Let's see how to cope with the first subgoal. The application of \texttt{K\_tac\_simple} leaves us with the subgoal

\begin{enumerate}
  \item \texttt{co\_l(M,F<-->co\_l(M,F))}
\end{enumerate}

We proceed using \texttt{co\_l\_ref\_tac} that applies *modus ponens* and \texttt{co\_l\_ref} and substitutes the old subgoal with the following two

\begin{enumerate}
  \item \texttt{F1<-->F<-->co\_l(M,F)}
\end{enumerate}
2. \( \text{co}_1(M, F) \)

The second subgoal can be resolved using axiom

\[
\text{DSL1} \quad "\text{co}_1(M, F<->\text{co}_1(M, F))"
\]

thus leading, after an application of the theorem

\[
\text{impI} \quad "(P<=>Q)<=>P-->Q"
\]

defined in \( \text{FOL} \) to handle the introduction of implication, to the subgoal

\[
1. \quad (F2<->\text{co}_1(M, F2))<=>\text{F}--\text{co}_1(M, F)
\]

that can be resolved using \text{IffE_tac}. Tactic \text{IffE_tac} is defined in \( \text{PC} \) to be used to prove subgoals that shape like \( A-->B \) trying to resolve the stronger \( A<->B \) or the equivalent \( B<->A \), the following if its code

\[
\text{fun IffE_tac(AXIOMS,n) = (}
\]

\[
((\text{resolve_tac [IffE_right] n})
\]

\[
\quad \text{THEN}
\]

\[
((\text{assume_tac n})
\]

\[
\quad \text{ORELSE}
\]

\[
((\text{resolve_tac AXIOMS n}))
\]

\[
\quad \text{ORELSE}
\]

\[
((\text{resolve_tac [IffE_left] n})
\]

\[
\quad \text{THEN}
\]

\[
((\text{assume_tac n})
\]

\[
\quad \text{ORELSE}
\]

\[
((\text{resolve_tac AXIOMS n}))
\]

\[
);\]

Once the theorem

\[
\text{IffE_right} \quad "F1<->F2<=>(F1-->F2)"
\]

defined in \( \text{PC} \) is applied, the double implication is first resolved against the available assumptions and then, if this fails, against the axioms provided as one of the arguments to the tactic. If the \( n \)th subgoal has not been reduced, the tactic tries the same strategy using this time theorem

\[
\text{IffE_left} \quad "F2<->F1<=>(F1-->F2)"
\]

instead of \text{IffE_right}. Subgoal \( \text{co}_1(M, \text{co}_1(M, F))-->\text{co}_1(M, F) \) is proved in the same way.

The following MaRK proof of \( D2 \) can be used as an example of the application of tactic \text{l_ref_tac}, moreover, it shows how to approach goals that have a conjunction in their consequences
Goal "F3<->(F1&F2)==>l(M,F3)==>(l(M,F1) & l(M,F2))";
by (resolve_tac [impI] 1);
by (resolve_tac [conjI] 1);
(* F3<->(F1&F2)==>l(M,F3)==>l(M,F1) *)
by (resolve_tac [mp] 1);
by (assume_tac 2);
back();
by (1_ref_tac(1));
by (eresolve_tac PC_RULES 1);
(* F3<->(F1&F2)==>l(M,F3)==>l(M,F1) *)
by (resolve_tac [mp] 1);
by (assume_tac 2);
back();
by (1_ref_tac(1));
by (eresolve_tac PC_RULES 1);
qed "D2";

The strategy is to split the conjunction and prove separately the subgoals obtained in this way. The first thing to do is to move \( l(M,F3) \) to the assumption by applying \( \text{impI} \); we get the following result

1. \[ [F3<->(F1&F2);l(M,F3)]==>(l(M,F1) & l(M,F2)) \]

Now it is possible to apply

\( \text{conjI} \) \( "[P;Q][]==>(P&Q)" \),

which is defined in \( \text{FOL} \) to handle the introduction of conjunction, and obtain the subgoals

1. \[ [F3<->(F1&F2);l(M,F3)]==>(l(M,F1) \]
2. \[ [F3<->(F1&F2);l(M,F3)]==>(l(M,F2) \]

Let's see how to prove the first subgoal, the second is proved in the same way. First we get back \( l(M,F3) \) from the assumptions to the consequence. We use \emph{modus ponens} that produces the subgoals

1. \[ [F3<->(F1&F2);l(M,F3)]==>(F==>l(M,F1) \]
2. \[ [F3<->(F1&F2);l(M,F3)]==>(F \]

then, we resolve the second subgoal using the assumptions. Tactic \( \text{assume_tac} \) tries to unify the consequence of a subgoal with one of its assumptions. In this case, a simple application of this tactic would unify \( F \) with the first assumption, but we need to unify it with the second one and for this reason we issue the command \( \text{back()} \). This command makes the theorem prover to back trace the application of \( \text{assume_tac} \) and pick \( l(M,F3) \) for the unification; this is the result we get

1. \[ F3<->(F1&F2)==>l(M,F3)==>(l(M,F1) \]
We can now apply \texttt{l\_ref\_tac} and reduce the subgoal to

1. F3\textless{}\Rightarrow(F1\&F2)\Rightarrow F3\Rightarrow F1

The proof is concluded applying the theorems defined in \texttt{PC}.

To conclude this section, we show that as we proved \texttt{l\_ref} and \texttt{co\_l\_ref} and used them to build proper tactics, we can use \texttt{D3}, once proved, to prove \texttt{D4}. First let’s see how the MaRK proof for \texttt{D3} proceeds, the following it’s the code of this proof

\begin{verbatim}
Goal"F3\textless{}\Rightarrow(F1\Rightarrow F2)\Rightarrow co\_l(M,F3)\Rightarrow l(M,F1)\Rightarrow l(M,F2)"
by (resolve_tac [impI] 1);
by (l2co\_l\_tac(1));
by (K\_tac\_simple(1));
by (co\_l\_ref\_tac(1));
by (assume\_tac 2);
by (eresolve\_tac PC\_RULES 1);
qed "D3";
\end{verbatim}

We state the goal to be proved and then we move \texttt{co\_l(M,F3)} to the assumptions exploiting \texttt{impI}. We have now the subgoal

\[ [[F3\textless{}\Rightarrow(F1\Rightarrow F2); co\_l(M,F3)]]\Rightarrow l(M,F1)\Rightarrow l(M,F2) \]

Using \texttt{l2co\_l\_tac} we can rewrite the consequence of the subgoal to obtain

\[ [[F3\textless{}\Rightarrow(F1\Rightarrow F2); co\_l(M,F3)]]\Rightarrow co\_l(M,\neg F2)\Rightarrow co\_l(M,\neg F1) \]

At this point an application of \texttt{K\_tac\_simple} produces

\[ [[F3\textless{}\Rightarrow(F1\Rightarrow F2); co\_l(M,F3)]]\Rightarrow co\_l(M,\neg F2\Rightarrow \neg F1) \]

and \texttt{co\_l\_ref\_tac} splits this subgoal in the following

1. \[ [[F3\textless{}\Rightarrow(F1\Rightarrow F2); co\_l(M,F3)]]\Rightarrow F4\Rightarrow (\neg F2\Rightarrow \neg F1) \]
2. \[ [[F3\textless{}\Rightarrow(F1\Rightarrow F2); co\_l(M,F3)]]\Rightarrow co\_l(M,F4) \]

Unifying the consequence of the second subgoal with the second assumption, we get

\[ [[F3\textless{}\Rightarrow(F1\Rightarrow F2); co\_l(M,F3)]]\Rightarrow F3\Rightarrow (\neg F2\Rightarrow \neg F1) \]

that can be proved exploiting the theorems defined in \texttt{PC}.

Axiom \texttt{D4} is now easy to prove, indeed it is an instance of \texttt{D3} and consequently the only steps we have to perform in order to build its proof are
to apply D3 and cope with the subgoal \( F \leftarrow (\text{True} \rightarrow F) \) that can be proved by the Isabelle simplifier. The following is the code of the proof

Goal "\( \text{co}_1(M,F) \rightarrow (l(M, \text{True}) \rightarrow l(M,F)) \)";
by (resolve_tac [D3] 1);
by (Simp_tac 1);
qed "D4";
5.2.1.3 dDSTL: coding ΔDSTL(x)

As ΔDSTL(x) has been built by adding a temporal structure on top of DSTL(x), theory dDSTL extends DSTL and introduces the definition of all the temporal operators together with their rules. Table 7 shows an extract of the DSTL.thy file where the coding of the temporal operators and of some of the rules is shown.

```
(* dDSTL: Distributed State Temporal Logic *)
dDSTL = DSL +
consts
(* event operator *)
delta :: o => o
(* dstl operators *)
leads_to :: [o,o] => o ("_LEADSTO_")
leads_to_c :: [o,o] => o ("_LEADSTOC_")
init :: o => o ("INIT_")
stable :: o => o ("STABLE _")
...
rules
(* introduction and elimination rules *)
LI "F1 LEADS_TO C F2===>F1 LEADS_TO F2"
InI "F===>INIT F"
SI "F===>STABLE F "
deltaFB F LEADS_TO G==>({F&-G) LEADS_TO delta(G)}
...
(* transitivity rules *)
LTR "[F1 LEADS_TO F2;F2 LEADS_TO F3]==>F1 LEADS_TO F3"
BTR "[F1 BECAUSE F2;F2 BECAUSE F3]==>F1 BECAUSE F3"
...
(* premises and consequences strengthening and weakening *)
LSW "[[G1-->F1;F1 LEADS_TO F2;F2-->G2]]===>G1 LEADS_TO G2"
LDF "[[F1 LEADS_TO G;F2 LEADS_TO G]]==>{F1|F2) LEADS_TO G"
LLC "[[G LEADS_TO F1;G LEADS_TO F2]]==>{G LEADS_TO (F1&F2)}"
(* notification *)
Notif "[[F BECAUSE G1;G1 LEADS_TO 1(M,G2)];STABLE{1(M,G2)}]]==>
(F&1(M, True)) LEADS_TO 3(M,G2)"
(* confluence *)
Conf "[[STABLE{1(M,F1)};STABLE{1(M,F2)}]];]==>{1(M,F1) & 1(M,F2)}==>{1(M,F1&F2)}"
(* properties of the initial state *)
II "INIT 1(M,True)"
IW "[[INIT F ;F--G]];===>INIT G"
IC "[[INIT F ;INIT F1]];===>INIT (F&F1)"
...
end
```

Table 7. Theory dDSTL.

To increase the readability of specifications and the usability of MaRK, besides the abstract syntax, a concrete syntax has been defined for the temporal operators and has been used in the rules of this theory. Consider, for example, the constructor leads_to used to represent operator
$LEADS\_TO$. It that has two DSL formulae as arguments and a dDSTL formula as result; in its abstract syntax, a formula built using this operator shapes like $leads\_to(F1,F2)$ while using the concrete infix syntax definition the same formula is written as $F1 \ LEASTO \ F2$, which definitely recalls the syntax presented in Section 4.2.2.

## 5.2.1.4 dDSTL tactics and proofs

Using MaRK, we proved several theorems that have been exploited during the proofs of the Mob$_{adtl}$ properties in Section 5.2.2 and of the properties of the examples in Section 6. This section reports the proofs of some of these theorems; for the complete code of the proofs of all the theorems see Appendix B.

Among the ΔDSTL(x) theorems that we proved there are $Cor1$ and $Cor2$ that have been widely used in the proofs for Mob$_{adtl}$ systems properties performed so far in MaRK and that are based on premises/consequences strengthening/weakening and transitivity of operator $LEADS\_TO$. Similar theorems have been proved for operator $BECAUSE$.

$$
\begin{align*}
\text{Cor1} & \quad F \text{ LEADS TO } G \land F' \implies G \text{ LEADS TO } F\\
\text{Cor2} & \quad F \text{ LEADS TO } G \land F' \implies G \text{ LEADS TO } F'
\end{align*}
$$

The proofs of these two theorems are similar, and we describe in detail the one of $Cor1$. The first step has been to identify a more general version of this theorem that shapes

$$
\begin{align*}
F \text{ LEADS TO } F1.1 \lor F2.1 \\
F1.1 \to F1.3 \quad F1.3 \text{ LEADS TO } F2.3 \quad F2.3 \to F3 \\
F2.1 \to F1.2 \quad F1.2 \text{ LEADS TO } F2.2 \quad F2.2 \to F3 \\
F \text{ LEADS TO } F3
\end{align*}
$$

and can be constructed joining theorems LPD, LSW and LTR as show in the following line of code from dDSTL where the theorem $ger\_Cor1$ is defined

```ml
val ger\_Cor1 = (LSW RS (LSW RSN (2,LPD))) RSN (2,LTR);
```

This theorem is built using Isabelle operators RS and RSN: operator RS is used to join two theorems using the second to resolve the first assumption of the first; operator RSN is used to join two theorems using the second to resolve the n$^{th}$ assumption of the first. To build $ger\_Cor1$ we proceed as follows: first, we resolve the second assumption of LPD using LSW, and then we use LSW to resolve the first assumption of the theorem obtained so far. The last step is to use the theorem obtained so far to resolve the second assumption of LTR. This is the schema of this construction
Once we have defined \texttt{ger\_Cor1}, we can exploit it to prove \textit{Cor1} that in MaRK gets the name \texttt{Cor1\_1}; the following is the proof.

Goal 

\[
\begin{align*}
\text{Goal} & \quad \text{"[|F1 \text{ LEADSTO } (G1|G2); G1 \text{ LEADSTO } F2|] => F1 \text{ LEADSTO } (F2|G2)";} \\
& \quad \text{by (resolve_tac [ger\_Cor1] 1);} \\
& \quad \text{by (assume_tac 1);} \\
& \quad \text{by (resolve_tac [Imp\_ref1] 4);} \\
& \quad \text{by (resolve_tac [LI\_1] 4);} \\
& \quad \text{by (Simp\_tac 4);} \\
& \quad \text{by (assume_tac 2);} \\
& \quad \text{back();} \\
& \quad \text{by (Simp\_tac 1);} \\
& \quad \text{by (Simp\_tac 1);} \\
& \text{qed "Cor1\_1";} 
\end{align*}
\]

After the application of \texttt{ger\_Cor1}, the goal is substituted by the following subgoals that correspond to the assumptions of \texttt{ger\_Cor1}.

1. \[[| F1 \text{ LEADSTO } G1 \mid G2; G1 \text{ LEADSTO } F2 |] \Rightarrow F1 \text{ LEADSTO } F1.1 \mid F2.1
2. \[[| F1 \text{ LEADSTO } G1 \mid G2; G1 \text{ LEADSTO } F2 |] \Rightarrow F1.1 \Rightarrow F1.3
3. \[[| F1 \text{ LEADSTO } G1 \mid G2; G1 \text{ LEADSTO } F2 |] \Rightarrow F1.3 \text{ LEADSTO } F2.3
4. \[[| F1 \text{ LEADSTO } G1 \mid G2; G1 \text{ LEADSTO } F2 |] \Rightarrow F2.3 \Rightarrow F2 \mid G2
5. \[[| F1 \text{ LEADSTO } G1 \mid G2; G1 \text{ LEADSTO } F2 |] \Rightarrow F2.1 \Rightarrow F1.2
6. \[[| F1 \text{ LEADSTO } G1 \mid G2; G1 \text{ LEADSTO } F2 |] \Rightarrow F1.2 \text{ LEADSTO } F2.2
7. \[[| F1 \text{ LEADSTO } G1 \mid G2; G1 \text{ LEADSTO } F2 |] \Rightarrow F2.2 \Rightarrow F2 \mid G2

We can prove the first subgoal resolving the consequence against the first assumption thus unifying \texttt{F1.1} with \texttt{G1} and \texttt{F2.1} with \texttt{G2}.

1. \[[| F1 \text{ LEADSTO } G1 \mid G2; G1 \text{ LEADSTO } F2 |] \Rightarrow G1 \Rightarrow F1.3
2. \[[| F1 \text{ LEADSTO } G1 \mid G2; G1 \text{ LEADSTO } F2 |] \Rightarrow F1.3 \text{ LEADSTO } F2.3
3. \[ | [ [ F1 \text{ LEADSTO } G1 | G2; G1 \text{ LEADSTO } F2 ] ] \] 
\[ \Rightarrow F2.3 \rightarrow F2 | G2 \]
4. \[ | [ [ F1 \text{ LEADSTO } G1 | G2; G1 \text{ LEADSTO } F2 ] ] \] 
\[ \Rightarrow G2 \rightarrow F1.2 \]
5. \[ | [ [ F1 \text{ LEADSTO } G1 | G2; G1 \text{ LEADSTO } F2 ] ] \] 
\[ \Rightarrow F1.2 \text{ LEADSTO } F2.2 \]
6. \[ | [ [ F1 \text{ LEADSTO } G1 | G2; G1 \text{ LEADSTO } F2 ] ] \] 
\[ \Rightarrow F2.2 \rightarrow F2 | G2 \]

The fourth subgoal can now be proved using Impl_ref1 getting the unification of F1.2 with G2.

1. \[ | [ [ F1 \text{ LEADSTO } G1 | G2; G1 \text{ LEADSTO } F2 ] ] \] 
\[ \Rightarrow G1 \rightarrow F1.3 \]
2. \[ | [ [ F1 \text{ LEADSTO } G1 | G2; G1 \text{ LEADSTO } F2 ] ] \] 
\[ \Rightarrow F1.3 \text{ LEADSTO } F2.3 \]
3. \[ | [ [ F1 \text{ LEADSTO } G1 | G2; G1 \text{ LEADSTO } F2 ] ] \] 
\[ \Rightarrow F2.3 \rightarrow F2 | G2 \]
4. \[ | [ [ F1 \text{ LEADSTO } G1 | G2; G1 \text{ LEADSTO } F2 ] ] \] 
\[ \Rightarrow G2 \rightarrow F2 | G2 \]

An application of LI and LcI permit to prove the fourth subgoal identifying F2.2 with G2.

1. \[ | [ [ F1 \text{ LEADSTO } G1 | G2; G1 \text{ LEADSTO } F2 ] ] \] 
\[ \Rightarrow G1 \rightarrow F1.3 \]
2. \[ | [ [ F1 \text{ LEADSTO } G1 | G2; G1 \text{ LEADSTO } F2 ] ] \] 
\[ \Rightarrow F1.3 \text{ LEADSTO } F2.3 \]
3. \[ | [ [ F1 \text{ LEADSTO } G1 | G2; G1 \text{ LEADSTO } F2 ] ] \] 
\[ \Rightarrow F2.3 \rightarrow F2 | G2 \]
4. \[ | [ [ F1 \text{ LEADSTO } G1 | G2; G1 \text{ LEADSTO } F2 ] ] \] 
\[ \Rightarrow G2 \rightarrow F2 | G2 \]

The fourth goal can be proved by a call to Simp_tac and the second one can be proved resolving the consequence against the second assumption, via the calls to assume_tac and back, unifying F1.3 with G1 and F2.3 with F2. This leaves us with the following subgoals that can be proved by the simplifier:

1. \[ | [ [ F1 \text{ LEADSTO } G1 | G2; G1 \text{ LEADSTO } F2 ] ] \] 
\[ \Rightarrow G1 \rightarrow G1 \]
2. \[ | [ [ F1 \text{ LEADSTO } G1 | G2; G1 \text{ LEADSTO } F2 ] ] \] 
\[ \Rightarrow F2 \rightarrow F2 | G2 \]

This proof is an example of a general strategy to prove a theorem. This strategy prescribes to look for more general version of the theorem that can be exploited to simplify the proof. Indeed, the proof of Cor1_l has been
reduced just to the instantiation of the assumptions of ger_Cor1 to make them match the particular case of Cor1_1. The same strategy has been applied to prove Cor2 and variants of Cor1 and Cor2 and to prove their BECAUSE counterparts; for the complete code see Appendix B.

The next proofs show how to use some theorems of the operator LEADS_TO; similar strategies can be applied to cope with goals built using operator BECAUSE.

Theorem DSTLlemma1 allows moving the premises of a LEADS_TO based formula to its consequences. The idea is that if a distributed state that satisfies the consequences of the formula exists, then, it is possible to build a distributed state that satisfies also the premises by extending the former to include the state in which the premises hold.

Goal "\(1(M,F1) \text{ LEADSTO } l(N,F2)\)\(\Rightarrow 1(M,F1) \text{ LEADSTO } (1(M,F1) \& l(N,F2))\); by (resolve_tac [LCC] 1); by (resolve_tac [LI] 1); by (resolve_tac [LcI] 1); by (Simp_tac 1); qed "DSTLlemma1";

Once stated the goal, an application of

\[
\text{LCC } \{ \text{G LEADS_TO F1; G LEADS_TO F2}\}\Rightarrow \text{G LEADS_TO (F1&F2)}
\]

reduces it to

1. \(1(M,F1) \text{ LEADSTO } l(N,F2)\)\(\Rightarrow 1(M,F1) \text{ LEADSTO } l(M,F1)\)
2. \(1(M,F1) \text{ LEADSTO } l(N,F2)\)\(\Rightarrow 1(M,F1) \text{ LEADSTO } l(M,F2)\)

The first subgoal can be dealt with using theorems

\[
\text{LI } "F1 \text{ LEADS_TO_C F2} \Rightarrow F1 \text{ LEADS_TO F2}"
\]
\[
\text{LcI } "l(M,F) \text{ LEADSTOC } l(M,F)"
\]

defined in dDSTL. The first theorem permits to reduce the subgoal to

\(1(M,F1) \text{ LEADSTO } l(N,F2)\)\(\Rightarrow 1(M,F1) \text{ LEADSTOC } l(M,F1)\)

whose consequence shapes like LcI that can be used to prove this subgoal. The consequence of the second subgoal matches exactly with the assumption and the simplifier invoked by a call to Simp_tac can prove it.

Theorem DSTLlemma2 is based on LSW and is a particular case of premises weakening: the premise is a conjunction of two localized formulae and the consequence if one of the two formulae. This is the code of the proof

Goal "\((1(M,F1) \& l(N,F2)) \text{ LEADSTO } l(M,F1)\)";
by (resolve_tac [LSW] 1);
by (resolve_tac [lemma13] 1);
by (resolve_tac [LI] 1);
by (resolve_tac [LcI] 1);
by (Simp_tac 1);
qed "DSTLlemma2";

The proof is pretty straightforward. First, we resolve the mail goal by applying

\[
\text{LSW } \left[ | G1 \rightarrow F1; F1 \text{ LEADS TO } F2; F2 \rightarrow G2 | \right] \Rightarrow G1 \text{ LEADS TO } G2
\]

and obtain the subgoals

1. \( (l(M,F1) \& l(N,F2)) \Rightarrow F3 \)
2. \( F3 \text{ LEADSTO } F4 \)
3. \( F4 \rightarrow l(M,F1) \)

then we proceed by exploiting

\[
\text{lemma13 } (A \& B) \Rightarrow A
\]

defined in PC to resolve the first subgoal and unify \( F3 \) with \( l(M,F1) \) to get

1. \( l(M,F1) \text{ LEADSTO } F4 \)
2. \( F4 \rightarrow l(M,F1) \)

An application of \( LI \) and \( LcI \) allows resolving the first subgoal and instantiate \( F4 \) to \( l(M,F1) \) leaving the following implication to prove

\( l(M,F1) \Rightarrow l(M,F1) \)

A call to the simplifier concludes the proof.

Theorem \( \text{DSTLlemma4} \) is another example of the applications of \( \text{LSW} \); in this case we exploit consequences strengthening.

Goal "\( F1 \text{ LEADSTO } (F1|F2) \)"
by (resolve_tac [LSW] 1);
by (resolve_tac [Imp_refl] 1);
by (resolve_tac [LI] 1);
by (resolve_tac [LcI] 1);
by (Simp_tac 1);
qed "DSTLlemma4";

The pattern of the proof is the same as for \( \text{DSTLlemma2} \). An application of \( \text{LSW} \) gives

1. \( F1 \Rightarrow F3 \)
2. \( F3 \text{ LEADSTO } F4 \)
3. \( F4 \Rightarrow (F1|F2) \)
We can use Imp_refl "A-->A" defined in PC to unify F3 with F1 and obtain

1. F1 LEADSTO F4
2. F4--> (F1|F2)

Then we can apply LI and LcI and resolve the first subogal unifying F4 with F1 and getting the implication

1. F4--> (F1|F2)

that can be proved by the simplifier.
5.2.2 Coding the Mob\textsubscript{adtl} theories

The hierarchy of the theories that globally represent the extension of the Isabelle generic theorem prover to support the reasoning on Mob\textsubscript{adtl} systems is presented in Figure 4. In the figure, the plain arrows stand for the extension/inclusion relation existing between the theories, and the dotted arrows highlight the refinement relation that connect the theories coding the Mob\textsubscript{adtl} reference model with the theory that contains the coordination rules of the Mob\textsubscript{adtl} model requirements. The structure of the theories in the figure follows faithfully the one presented in Section 4.3 for the refinement process that is at the basis of our methodology.

![Figure 4. Model theories hierarchy](image)

At the top of the hierarchy, as the starting point of the process that leads to the definition of the Isabelle theories for Mob\textsubscript{adtl} reference model, there is the theory \texttt{mob\_adtl\_support} that defines everything is needed by the other model theories as reported in Table 8. All the predicates used in the axiomatisation of the model requirements and the reference model are defined in this theory, moreover a proper structure is given to lists of component names and to profiles. Type \texttt{component\_name\_list} is declared and constants \texttt{nilCNL} and \texttt{consCNL} are introduced, the former being the empty list of component names and the latter being the constructor for this type of lists; this means that \texttt{consCNL(M,nilCNL)} is the singleton list containing the name \texttt{M}. Constants \texttt{subCNL}, \texttt{isInCNL} and \texttt{eqCNL} are defined to model the sublist relation, the belonging of a component name to a list and the equality of component names lists respectively. In the rules section, rules to reason about these relations are given. Rule \texttt{isInCNL\_rule} says that a component name \texttt{C1} belongs to a list that results from the composition of the component name \texttt{C2} and of the list \texttt{L1} if \texttt{C1} and \texttt{C2} are the same name or \texttt{C1} belongs to \texttt{L1}. Rule \texttt{subCNL\_nil} says that the empty list is sublist of any list. Rule \texttt{subCNL\_rule} exploits \texttt{isInCNL\_rule} and \texttt{subCNL\_nil} to check,
given two lists, if all the component names contained in the first are
contained also in the second.

Profiles, i.e. items of type profile, are built starting from elements of
the type profile_item that models the single constraints that constitute a
profile. Constructors similar to those defined for component names lists are
introduced also for profiles, and constants to model the belonging of a profile
item to a profile (isInP) and the implication relation between two profiles
(impliesP) are given as well. As consCNL(M,nilCNL) is the singleton list
containing the name M, consP(P,nilP) is the profile that contains only the
constraint P.

Proper types are defined to model the state of a mobility or
communication request (req_state) and to describe exception types and
exception data (exc_type and exc_data respectively). Type req_state is
the type of constant i and u that are used for unresolved requests and
mobility requests in their initial state, and are the result type of constructors
rec and dest that get a component name as argument. Type exc_type is
the type of constructors msg and mob that models communication and
mobility requests in exceptions respectively. The result of these constructors
is used togethether with an element of type exc_data, e.g. dr, by toBeVetoed
and exc to model vetoes in the state of guardians and components
respectively.

The constructors related to mobility requests are moveTo, which gets as
arguments a profile and a component names list to model the history, and
mobReq, which gets as arguments the name of the component that issued the
request, the name of its origin guardian, a profile and the state of the request.

Messages are given the type message that is used by out and msgReq to
model the will of communicate of an agent and the communication request
that is received by the guardians. There are not constructors for messages,
since at this level, there is no need to fix a structure for them; any refinement
of this reference model can introduce all the needed constructors and define
different handling rules for messages with different forms.

Constructors guarding, guardedby and history are defined as
natural given the types presented so far.
(* everything is needed by mob_adtl models: types, structures, predicates...*)

mob_adtl_support = dDSTL +

    types
    
    component_name_list
    profile_item
    profile
    message
    req_state
    exc_type
    exc_data

    ...consts

    nilCNL :: component_name_list
    consCNL :: [component_name_list, component_name_list] => component_name_list
    subCNL :: [component_name_list, component_name_list] => 0
    isInCNL :: [component_name, component_name_list] => 0
    eqCNL :: [component_name_list, component_name_list] => 0

    nilP :: profile
    consP :: [profile_item, profile] => profile
    impliesP :: [profile, profile] => 0
    isInP :: [profile_item, profile] => 0

    ...guardedby :: component_name => 0
    moveTo :: [profile, component_name_list] => 0
    history :: component_name_list => 0
    guarding :: component_name => 0
    moving :: component_name => 0
    mobReq :: [component_name, component_name, profile, req_state] => 0
    msgReq :: [message, component_name, profile, req_state] => 0
    msg :: [message, component_name, profile] => exc_type
    mob :: [component_name, component_name, profile] => exc_type
    exc :: [exc_type, exc_data] => 0
    toBeVetoed :: [exc_type, exc_data] => 0
    in :: [message, component_name] => 0
    out :: [message, profile] => 0
    i :: req_state
    u :: req_state
    dest :: component_name => req_state
    rec :: component_name => req_state
    dr :: exc_data

    ...

    rules

    isInCNL_rule "[(C1=C2)|isInCNL(C1,L)|] => isInCNL(C1,consCNL(C2,L))"
    subCNL.nil "subCNL(nilCNL,L)"
    subCNL_rule "[(isInCNL(C,L2)|subCNL(L1,L2)|] => subCNL(consCNL(C,L1),L2)"
    eqCNL_rule "=subCNL(L2, consCNL(C, L2))"

End

Table 8. Support theory for Mobadtl Isabelle theories.
Once the constructors that model all the predicates involved in the formalisation of the Mob\textsubscript{adtl} model are given, the Isabelle coding of the theories for the model requirements and the reference model is pretty straightforward. A Isabelle theory corresponds to each Mob\textsubscript{adtl} theory and the extension relation holding for theories of different levels is implemented in the inclusion relation defined for Isabelle theories. For example, the following tables show the Isabelle theories for Mob\textsubscript{adtl} theories \textit{A0} and \textit{A1} from requirements specification and reference model respectively.

\begin{table}[h]
\centering
\begin{tabular}{|l|}
\hline
\texttt{(* mob\_adtl requirements specification: theory A0, agents' behaviour *)} \\
\texttt{mob\_adtl\_A0 = dDSTL + mob\_adtl\_support +} \\
\texttt{rules} \\
\texttt{U "ALL A G1 G2 . co\_1(A,guardedby(G1)&guardedby(G2)--->eq(G1,G2))"} \\
\texttt{S1 "ALL A P H . 1(A,m\_moveTo(P,H)--->history(H))"} \\
\texttt{S2 "ALL A G1 H . EX G2 . 1(A,guardedby(G1)&history(H)) UNLESS} \\
\texttt{1(A,guardedby(G2)&history(consCNL(G1,H)))"} \\
\texttt{S3 "ALL A G H . EX P . 1(A,history(consCNL(G,H))) BECAUSE 1(A,m\_moveTo(P,H))"} \\
\texttt{S4 "ALL A . INIT 1(A,history(nilCNL))"} \\
\texttt{S5 "ALL A . co\_1(A, EX G H . guardedby(G) & history(H))"} \\
\hline
\end{tabular}
\caption{Theory mob\_adtl\_A0, requirements specification for the agents.}
\end{table}

\begin{table}[h]
\centering
\begin{tabular}{|l|}
\hline
\texttt{(* mob\_adtl reference model: theory A1, agents' behaviour *)} \\
\texttt{mob\_adtl\_A1 = mob\_adtl\_A0 +} \\
\texttt{rules} \\
\texttt{Ia "ALL A P1 H M P2 Type D . INIT 1(A,-m\_moveTo(P1,H)&-out(M,P2)&-exc(Type,D))"} \\
\hline
\end{tabular}
\caption{Theory mob\_adtl\_A1, reference model for the agents.}
\end{table}

Theory \textit{mob\_adtl\_A0}, which is the Isabelle version of theory \textit{A0}, extends dDSTL importing all the definitions contained in \textit{mob\_adtl\_support}; on its turn, \textit{mob\_adtl\_A1} codes theory \textit{A1} extending \textit{mob\_adtl\_A0} and defining a new axiom.

Note that in their Isabelle version, all the axioms are explicitely quantified. The quantification of MaRK theories faithfully reflects the implicit quantification given for \textit{ADSTL(x)}. During the proofs, quantifiers must be dealt with exploiting ordinary introduction and elimination rules defined for first order quantification. Some examples are shown in Section 5.2.3.1.
5.2.3 Mob$_{addl}$ proofs

Recall that, as said in Section 4.3, given a refinement chain, in order to be correct, the $i^{th}$ refinement step must satisfy the following conditions:

1. theory $A_i$ refines $A_{i-1}$
2. $G_i$ refines $G_{i-1}$
3. $A_i \cup G_i \cup C_i$ refines $C_{i-1}$.

where the refinement relation is the logic implication among $\Delta$DSTL(x) theories. To prove the reference model correct we must instantiate this conditions on theories of level 0 and level 1, the requirements specification and the reference model of Mob$_{addl}$ respectively.

To perform this task we used MaRK. MaRK theories mob$_{adtl\_A1}$ and mob$_{adtl\_G1}$ satisfy the first and the second condition respectively by construction, since they include their counterparts of level 0, as explained in Section 5.2.2. What we need to prove is thus that the union of the MaRK theories of level 1 allows deriving all the properties specified in theory mob$_{adtl\_C0}$. The three MaRK theories of level 1 are collected in theory mob$_{adtl\_reference\_model}$, consequently we must be able to performs all the needed proofs using this theory.

The main characteristics of these proofs are the need of coping with quantifications, the acquisition of assumptions and the wide use of tactics. The next sections discuss these topics in details and present some of the proofs; for the complete code of all the proofs, see Appendix B.

5.2.3.1 Handling quantifiers and acquiring assumptions

All the proofs performed to check the correctness of theory mob$_{adtl\_reference\_model}$ are characterized by the need of handling the quantifiers: in order to apply DSTL and dDSTL theorems it is necessary to remove all the quantifiers and substitute quantified variables with proper parameters and skolems functions of these parameters. Isabelle implementation of first order logic FOL provides theorems that can be exploited to cope with the proof of quantified formulae. Given a goal with possibly quantified assumptions and consequence, theorems

allI "(\!x. P(x)) ==> ALL x. P(x)"
exI "P(?x) ==> EX x. P(x)"

can be used to remove quantifiers in the consequence, while theorems

spec "ALL x. P(x) ==> P(?x)"
exE "[| EX x. P(x); \!x. P(x) ==> R |] ==> R"

can be used to achieve the same result for the quantifiers in the assumptions. All these theorems are designed to remove one quantification per application; a repeated application will remove multiple quantifiers. Using allI to
resolve the consequence of a goal, one achieves to remove the ALL quantifier and creates a parameter, that’s the meaning of !!x, that will be used to build skolem functions like in exI, where the quantified variable x is substituted by ?x that is a function of all the available parameters. These two theorems are used in conjunction of resolve_tac like in the following commands

    by (resolve_tac [allI] 1);
    by (resolve_tac [exI] 1);

The first command matched the consequence of to the first subgoal with the consequence of allI and substitutes the subgoal with a ne one that has the same assumptions of the old one and as consequence the premises of allI properly instantiated.

The theorems designed to remove the quantifications from the assumptions of a goal work dually respect to allI and exI. Theorem spec is used in conjunction of tactic dresolve_tac that, as said in Section 5.1.2, works forward on a subgoal’s assumption. This means that when the command

    by (dresolve_tac [spec] 1);

is issued, an assumption of the first subgoal is matched with the premise of spec and substituted with the instantiation of the consequence of spec; the result is that the ALL quantifier is removed and the quantified variable is replaced by a skolem function of all the available parameters. Theorem exE is used in conjunction with tactic eresolve_tac that, as said in section 5.1.2, works like an application of resolve_tac followed by an application of assume_tac. The command

    by (eresolve_tac [exE] 1);

matches the consequence of the first subgoal with the consequence of exE and substitute the subgoal with two new subgoals that have the assumptions of the old subgoal and as consequence one of the consequences of exE respectively. Then the first ne subgoal is resolved by assumption, so to instantiate P(x) to the desidered term and to leave just one subgoal that is the old subgoal without the existential quantifier and with a parameter derived from the quantified variable.

When dealing with quantifiers it is important to remove them in the right order. The universal quantifiers in the consequence must be removed first so to create parameters for the skolem functions that will originate from the elimination of universal quantifiers in the assumptions and existential quantifiers in the consequence. Once quantifiers have been removed, dDSTL theorems can be used to reduce the goal.

During the proof of a goal can be necessary to acquire already proved theorems as assumptions of the goal. To this aim Isabelle users are provided with tactic cut_facts_tac. A call like
by (cut_facts_tac [Cm2']) 1

introduces an assumption based on axiom Cm2' among the assumptions of the first subgoal. When introducing quantified assumptions, it is important to pay the proper attention to the elimination of their quantifiers. The elimination of the quantifiers occurring in the new assumptions could indeed create parameters on which the skolem functions originated from the elimination of the quantifiers that occur in the goal should depend on. For this reason, the general pattern to manage the elimination of quantifiers when acquiring quantified assumptions is the following:

1. Remove universal quantifiers from the consequence of the goal to be proved (allI)
2. Introduce the assumptions one by one (cut_facts_tac) removing their universal and existential quantifiers in this order (spec,exE)
3. Remove the existential quantifiers from the consequence of the goal (exI)

Another subtlety that influences the correctness of this process is the order in which assumptions are acquired, indeed, one must be sure to base the skolem functions created by exI and spec on the proper parameters. Typically, in a transitivity-based proof, performed for example exploiting the theorem

LTR "[|F1 LEADS_TO F2;F2 LEADS_TO F3|]==>F1 LEADS_TO F3"

the universal quantifier of the second assumption can be removed only after the existential quantifier of the first assumption has been removed. This, indeed, allows substituting the universally quantified variables in the second assumption with skolem functions based on the parameters obtained from the existentially quantified variables in the first assumptions and makes it possible to match the consequences of the first assumption with the premises of the second one.

Following the steps defined above and paying the proper attention to the order of assumptions acquisition, one can be sure that when a skolem function is created it depends on the right parameters.

As an example of this process, let’s examine the proof of property M1 from the proof of correctness of mob_adtl_reference_model.
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Gm2'] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Gm1'] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (REPEAT (eresolve_tac [exI] 1));

... by (SW_tac_asm(BSW,[l_ref,lemma13],[Imp_ref1],1));
by (SW_tac_asm(BSW,[l_ref,lemma13],[l_ref,lemma13],1));
qed "M1"

Once the goal has been stated, the sequence of proof steps begins with

by (REPEAT (resolve_tac [allI] 1));

that removes the universal quantifier ALL A T by repeating until possible, i.e. twice, the application of allI; this step is followed by five groups each one of three steps like the following

by (cut_facts_tac [Cm2'] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));

These five triplets correspond to the acquisition of assumptions based on
Cm2', Gm3_6', Gm2', Gm1', and Cm1' performed by tactic cut_facts_tac, and to the elimination of their quantifications performed by repeatedly applying spec and exE. This section of the proof is concluded by step

by (REPEAT (resolve_tac [exI] 1));

that removes the existential quantifier P H G1 O St G2 D. Once these steps have been performed, the proof of M1 can proceed with the application of proper axioms to reduce the goal to subgoals and prove them.

During a proof like the one of property M1, is most likely that the user
does not know in advance all the needed assumptions. Assumptions are
depicted one by one while the proof proceeds: if one of the subgoals matches
with one of the available axioms, the axiom can be introduced as an
assumption and used to resolve the subgoal.

5.2.3.2 Tactics and proofs

The proof of correctenes of mob_adtl_reference_model heavily
relies on tactics that can be used to prove ddSTL goals. The fact that these
tactics have been depicted while trying to prove the properties of the Mobadtl
reference model provides an example of how MaRK users can improve and customize this tool by storing the experience they gain while using it.

As an example of the use of tactics, consider the proof of property M1 reported in Section 5.2.3.1. This proof is concluded by two steps based on tactic SW_tac_asm that is defined as follows in dDSTL

```haskell
fun SW_tac_asm(SW_RULE,AXIOMS_FirstPrem,
               AXIOMS_ThirdPrem,n) = (
  (resolve_tac [SW_RULE] n) THEN
  (case AXIOMS_FirstPrem of
   A1::L1 => rec_res_tac(AXIOMS_FirstPrem,n) THEN
     ((assume_tac n) THEN
      rec_res_tac(AXIOMS_ThirdPrem,n)) |
   [] => ((assume_tac (n+1)) THEN
      rec_res_tac(AXIOMS_ThirdPrem,n+1))));
```

This tactic, which has been widely used in the proofs for the reference model, has been defined to ease the application of theorems based on premises/consequences strengthening/weakening of operators LEADS_TO and BECAUSE, namely LSW and BSw. The tactic applies the rule from the first argument substituting the subgoal it has been applied to with the three premises of the rule

```haskell
(resolve_tac [SW_RULE] n)
```

then, a test on the list from the second argument is performed

```haskell
  (case AXIOMS_FirstPrem of
   A1::L1 => ...
   [] => ...)
```

If the case A1::L1 applies, i.e. the list contains at least one element, list AXIOMS_FirstPrem is used to prove the first premise

```haskell
  rec_res_tac(AXIOMS_FirstPrem,n)
```

It is assumed that AXIOMS_FirstPrem is enough to accomplish this task. After this, the second and third premises are resolved, the former using assumptions, the latter using AXIOMS_ThirdPrem which is supposed to be enough to prove the third premise

```haskell
  (assume_tac n) THEN
  rec_res_tac(AXIOMS_ThirdPrem,n)
```

If list AXIOMS_FirstPrem is empty, the second and third premises are resolved while the first is left untouched

```haskell
  (assume_tac (n+1)) THEN
  rec_res_tac(AXIOMS_ThirdPrem,n+1)
```
As an example of the application of this tactic, consider the following command

\[
\text{by (SW\_tac\_asm(BSW, [l\_ref, lemma13], [Imp\_ref1], i))}
\]

It applies rule BSW to the first subgoal of the proof state substituting it with the three premises of BSW; then l\_ref and lemma13 are applied in this order to the first of the three subgoals proving it completely and leaving two subgoals to be proved. The first of these goals is resolved against the assumptions and the second is proved via an application of Imp\_ref1.

Tactic SW\_tac\_asm can be given an empty list for the second or third parameter. If both lists are empty, the tactic is equivalent to an application of the rule in the first parameter. A version that does not relies on assumptions has been defined as well, it is is called SW\_tac and has an extra argument for the list of axioms to be used to prove the second premise of the rule applied.

fun SW\_tac(SW\_RULE, AXIOMS\_FirstPrem, AXIOMS\_SecondPrem, AXIOMS\_ThirdPrem, n) = 
  case AXIOMS\_FirstPrem of
    A1::L1 => rec\_res\_tac(AXIOMS\_FirstPrem, n) THEN
      ((case AXIOMS\_SecondPrem of
        A2::L2 =>
          rec\_res\_tac(AXIOMS\_SecondPrem, n) THEN
          rec\_res\_tac(AXIOMS\_ThirdPrem, n) |
          [] => rec\_res\_tac(AXIOMS\_ThirdPrem, n+1)))) |
    [] => ((case AXIOMS\_SecondPrem of
      A3::L3 =>
        rec\_res\_tac(AXIOMS\_SecondPrem, n+1) THEN
        rec\_res\_tac(AXIOMS\_ThirdPrem, n+1) |
        [] => rec\_res\_tac(AXIOMS\_ThirdPrem, n+2)))));

Tactics SW\_tac\_asm and SW\_tac are based on tactic rec\_res\_tac that has been built to apply a sequence of theorems to the same goal.

fun rec\_res\_tac(L, n) = 
  case L of
    [] => all\_tac |
    A::L1 => (resolve\_tac [A] n) THEN rec\_res\_tac(L1,n) ;

This tactic calls itself recursively and iterates the resolution tactic for all the theorems contained in the list L. When the list is empty, the tactic all\_tac, which always succeeds leaving untouched the proof state, is applied to make rec\_res\_tac to return.

A pattern similar to the one applied to the design of SW\_tac and SW\_tac\_asm is at the bases of tactic Tran\_tac. Even in this case we have a tactic that can be used with properties written with operators LEADS\_TO
and \textit{BECAUSE}. The tactic applies the rule from the first argument (LTR or BTR) substituting the subgoal it has been applied to with the two premises of the rule. These two premises are resolved by applying the theorems from the second and third arguments respectively that, if not empty, are supposed to be enough to accomplish this task. A call to this tactic with empty lists provided as second and third arguments is equivalent to an application of the rule \textit{TR\_RULE}.

\begin{verbatim}
fun Tran_tac(TR\_RULE, AXIOMS\_FirstPrem,
      AXIOMS\_SecondPrem,n) = (n
   (resolve_tac [TR\_RULE] n) THEN
   (case AXIOMS\_FirstPrem of
      A::L => (rec_res_tac(AXIOMS\_FirstPrem,n)) THEN
               (rec_res_tac(AXIOMS\_SecondPrem,n)) |
      [] => (rec_res_tac(AXIOMS\_SecondPrem,n+1))));
\end{verbatim}

As examples of the successful application of the tactics presented so far, let's examine the proof of properties \textit{lemmaM1} and \textit{lemmaC2} from \textit{mob\_adtl\_refModel}; the skeleton of the proofs for these properties are reported in reported in Table 11 and Table 12 respectively.

Property \textit{lemmaM1} guarantees that mobility requests are served as expected giving place to a movement or to a veto. In the main goal, each term like \textit{delta(exc(mob(A,O,P),D1))}, stands for a veto that the agent can receive as a result of its mobility request, indeed, the request can be vetoed by: the origin guardian because it is a double request or because the guardian cannot or does not want to serve it, by the destination guardian, and by a guardian in the middle. Once hypotheses are introduced among the assumptions and quantifiers are removed with steps

\begin{verbatim}
   by (REPEAT(resolve_tac [allI] 1));
   ...
   by (REPEAT (resolve_tac [exI] 1));
\end{verbatim}

it is possible to proceed with the proof by splitting the main goal using \textit{Tran\_tac} to trace the steps that define the iter a mobility request must follow. The first step is the one that leads from the will to move of the agent \textit{A} to the delivery of the mobility request in the guardian \textit{O} that is guarding the agent \textit{A}. The request reaches the origin guardian when one of \textit{guarding(A)} and \textit{moving(A)} holds and this calls for a reasoning by cases that defines two branches of the proof. If the mobility request reaches the guardian when \textit{guarding(A)} holds, the guardian starts serving the request and either it forwards it to another guardian or vetoes it. If the origin guardian forwards the requests, the request will reach the destination guardian \textit{T} if no other guardian

\begin{verbatim}

111
\end{verbatim}
Goal "ALL A P H O . EX T D1 D2 D3 .

1(A,delta(moveTo(P,H))) guardsby(0)) LEADSTO
1(A,delta(guardedBy(T)))
   delta(exc(mob(A,O,P),D1)))
   delta(exc(mob(A,O,P),D2)))
   delta(exc(mob(A,O,P),D3)))
   delta(exc(mob(A,O,P),dr)))"

by (REPEAT(resolve_tac [all1]));
...
by (REPEAT [resolve_tac [eq1] 1]);
(* from the will to move of A to the mobility request in O exploiting relations
between guardedby and guarding we can deduce that (guarding(A)\|moving(A)) holds
in the state in which the request arrives*)
by (resolve_tac [LTR] 1);
...
by (SW_tac(BSW,[l_ref,lemma14],[[],[Imp_ref1],1]));
(* from the mobility request in 0 to the outcome of the request splitting the cases
on the condition (guarding(A)\|moving(A)) *)
by (SW_tac(LSW,[IffE_right,Or_l,IffSimm],[[],[Imp_ref1],1]));
(* the mobility request reaches the guardian when guarding(A) holds; the guardian
starts serving the request, either it forwards it to another guardian or
vetoes it *)
by (Tran_tac(LTR,[[],[],1]));
(* the guardian forwards the request and the destination guardian receives the
request or another guardian vetoes it*)
by (Tran_tac(LTR,[[],[],1]));
(* the destination guardian receives the request and accepts or vetoes it*)
by (Tran_tac(LTR,[[],[],1]));
(* the destination guardian accepts the request *)
by (SW_tac_asm(LSW,[Imp_ref1],[l_ref],1));
(* the destination guardian vetoes the request *)
by (SW_tac_asm(LSW,[Imp_ref1],[l_ref],1));
(* another guardian vetoes the request *)
by (SW_tac_asm(LSW,[Imp_ref1],[l_ref],1));
(* the guardian vetoes the requests *)
by (SW_tac_asm(LSW,[Imp_ref1],[l_ref],1));
(* the mobility request reaches the guardian when moving(A) holds and the guardian
vetoes it *)
by (Tran_tac(LTR,[[],[],1]));
by (SW_tac_asm(LSW,[Imp_ref1],[l_ref],1));
qed "lemmaM1";

Table 11. Applying tactics, proof of lemmaM1.

The destination guardian can accept the request and start guarding the agent A, or it can raise a veto. The origin guardian can veto the request; moreover, guardian O raises a veto if it receive A’s request when A is moving, i.e. when O is serving another mobility request from A. The code in Table 11 follows this structure, each application of Tran_tac correspond to a step in the iter of the mobility request, and each application of SW_tac corresponds to the resolution of one of the premises introduced by an application of Tran_tac.
Goal "ALL S M P . EX R D1 D2 D3 .
   1(S,delta(out(M,P))guardedby(G)) LEADSTO
   1(R,delta(in(M,S))
   1(S,delta(exc(msg(M,S,P),D1)))
   1(S,delta(exc(msg(M,S,P),D2)))
   1(S,delta(exc(msg(M,S,P),D3)))";
by (REPEAT (resolve_tac [allI] 1));
...
by (REPEAT (resolve_tac [exI] 1));
(* from the will to send a message of A to the communication request in O *)
by (Tran_tac(LTR,[],[],[]);(*Cc1*)
(* a guardian will receive the request and resolve the name of the
receiver or the first guardian will vetoes the request *)
by (Tran_tac(LTR,[],[],[]);(*Gc1*)
(* the guardian that resolves the address either guards the receiver or
forward the request to a guardian that's guarding the receiver *)
by (SW_tac(LSW,[],[],[],[Imp_ref1],[]));
(* the guardian that resolves the address to a guardian in the middle
receives a message or vetoes it *)
by (Tran_tac(LTR,[],[],[]);(*Cc2*)
(* the guardians that resolves the address delivers the message *)
by (SW_tac(LSW,[Imp_ref1],[LI,LcI],[]),[]);(*Cc3*)
(* the guardians that receives and forwards the message to a guardian that's guarding the receiver *)
by (Tran_tac(LTR,[],[],[]);(*Gc2*)
(* the guardians that's guarding the receiver either delivers the message
or vetoes it *)
by (SW_tac(LSW,[Imp_ref1],[LI,LcI],[]),[]);(*Cc2*)
(* the guardians that's guarding the receiver delivers the message *)
by (SW_tac(LSW,[Imp_ref1],[LI,LcI],[]),[]);(*Cc3*)
(* the first guardians vetoes the request *)
by (SW_tac(LSW,[Imp_ref1],[],[]),[]);(*Cc3*)
qed "lemmaC2";

Table 12. Applying tactics, proof of lemmaC2.

Property lemmaC2 guarantees that communication requests are
served as expected giving place the delivery of the sent message or to a veto.
In the main goal, each term like delta(exc(msg(M,S,P),D1)), stands for
a veto that the agent can receive as a result of its communication request,
indeed, the request can be vetoed by: the origin guardian because the
guardian cannot or does not want to serve it, by the guardian that controls
the receiver, and by a guardian in the middle. Once hypotheses are
introduced among the assumptions and quantifiers are removed with steps

by (REPEAT(resolve_tac [allI] 1));
...
by (REPEAT (resolve_tac [exI] 1));
it is possible to proceed with the proof by splitting the main goal using \texttt{Tran\_tac} to trace the steps that define the iter a communications request must follow. The first step is the one that leads from the will to send a message of the agent $S$ to the delivery of the communication request in the guardian $G$ that is guarding the sender. The guardian starts serving the request and either it forwards it to another guardian or vetoes it. If guardian $G$ forwards the requests, the request will reach a guardian that resolves the profile and finds a proper receiver. If the guardian that resolves the profile is guarding the chosen receiver, and if no veto is raised, the message is immediately delivered. Otherwise, a veto is raised and communicated to the sender, or the message is forwarded to a guardian that is guarding the chosen receiver and that, on its turn, will deliver the message or raise a veto for the communication request. The code in Table 12 follows this structure, each application of \texttt{Tran\_tac} correspond to a step in the iter of the communication request, and each application of \texttt{SW\_tac} corresponds to the resolution of one of the premises introduced by an application of \texttt{Tran\_tac}. 
6 Examples

The examples presented in this section have the manifold aim of showing how to use Mob_adtl to illustrate how it integrates well with existing technologies and how it meets its requirement of joining the specification of the different kinds of mobility in a single model. The examples are presented following an increasing grade of difficulty. The first examples, in their simplicity, provide a tutorial-like approach to the specification methodology that accompanies Mob_adtl. Moreover, they introduce a general schema that will be used for all the other examples too: the refinement activity starts from the definition of a support theory that extends the support theory of the Mob_adtl reference model and includes all the elements on which the refinement will build. Then, all the needed theories are defined following the pattern presented in Section 4.4. The last two examples present a more complicated structure and allow a better appreciation of the refinement process and of the composition of different refinements.

All the examples presented are based on the idea of defining new reference models for particular classes of systems, those exploiting unicast or multicast communications, those in which components are referred with their own name, those that exploit service discovery mechanisms, and those that mix different kinds of mobility. As the Mob_adtl reference model, the new reference models can be used as the starting point of the specifications process for any particular system instance of one, or more, of these classes.

6.1 Working with profile resolution

The profile resolution mechanism at the basis of Mob_adtl communication and mobility has been developed as a means of generality that allows accommodating in a single framework different ways of identifying the recipients of messages and the targets of movements. The following examples show how to refine the Mob_adtl Reference model to describe particular forms of communication and mobility exploiting the profile resolution mechanism.

6.1.1 Unicast and multicast communications

A first, pretty natural, refinement of the reference model is the one that introduces unicast and multicast communications. The axioms of the general model do not prescribe anything about the number of recipients a message is delivered to. Thus, these refinements could constitute the first step towards the description of any system.

The main element of this refinement is the profile resolution mechanism. We specify unicast and multicast communication introducing two profile constructors to identify the two communication forms, and providing axioms that describe how the guardians manage unicast and multicast message requests. Figure 5 shows the MaRK theories for these examples and their relationships.
The support theories extend directly `mob_adtl_support` and introduce the constructors used to build the terms that model the profile specification. Table 13 reports the support theory for multicast communications.

```
mob_adtl_multicast_support.thy
{*
   everything is needed by multicast theories *
}*

mob_adtl_multicast_support = mob_adtl_support +
  const
multi :: profile_item
End
```

Table 13. Multicast support theory.

The `multi` constructor is used to build communication requests as shown in the axioms from the theory that describes the management of multicast messages. Theory `mob_adtl_multicast_G` extends the `Mob_adtl` reference model and adds an axiom that models the delivery of a message to all the recipients that satisfy profile \( P \). Axiom `Multi` exploits the universal quantification on \( R \) to prescribe that a guardian that knows how to resolve the profile \( P \) delivers a multicast message sent to \( P \) to all the receivers that satisfy that profile, if a veto is not raised.

The refinement leading to the definition of unicast communications is slightly more complicated. The reason for this is that to provide unicast messages delivery it is necessary to ensure that only one guardian is called to resolve the profile to which the message is sent; otherwise, we cannot guarantee the uniqueness of the recipient. A possible solution to this problem is given by axiom `Unil` from theory `mob_adtl_unicast_C` that identifies in the guardian of the sender the one that resolves the profile. The `uni_profile` constructor is defined in theory `mob_adtl_unicast_support` that is analogous to the support theory for multicast.
mob_adtl_multicast_G.thy

(* specification of unicast communications in mob_adtl:
   guardians' behaviour *)
mob_adtl_multicast_G = mob_adtl_refModel + mob_adtl_multicast_support +
rules
Multil "ALL M S P R . EX D .
   1(G,delta{msgReq(M,S,consP{multi,P},u)}) &
   satisfy(consCNL(R,nilCFL),consP{multi,P})} LEADSTO
   1(G,delta{msgReq(M,S,consP{multi,P},rec(R))})
   1(G,toBeVetoed(msg(M,S,consP{multi,P}),D))"
end

Table 14. Multicast guardians theory.

mob_adtl_unicast_C.thy

(* specification of unicast communications in mob_adtl:
   coordination rules *)
mob_adtl_unicast_C = mob_adtl_refModel + mob_adtl_unicast_support +
rules
Unitl "ALL G M S P .
   1(G,delta{msgReq(M,S,consP{uni,P},u)}) BECAUSE
   1(S,delta{out(M,consP{uni,P})&guardedby(G)})"
end

Table 15. Unicast coordination theory.

On the guardians’ side, moreover, to guarantee the uniqueness of the
recipient, we must explicitly force profile resolution to give one agent as
result, as done by axioms Uni2 and Uni3 in theory mob_adtl_unicast_G.
The way this agent is chosen among the many that satisfy a profile it is not
shown, since it could depend on particular application requirements and
consequently it is not eligible to be part of a reference model. This is a hook
for further refinements.

mob_adtl_unicast_G.thy

(* specification of unicast communications in mob_adtl:
   guardians' behaviour *)
mob_adtl_unicast_G = mob_adtl_refModel + mob_adtl_unicast_support +
rules
Uni2 "ALL G M S P R1 R2 .
   1(G,msgReq(M,S,consP{uni,P},rec(R1))&
   msgReq(M,S,consP{uni,P},rec(R2))->eqCN(R1,R2))"

Uni3 "ALL G M S P R .
   STABLE(1(G,msgReq(M,S,consP{uni,P},rec(R))))"
end

Table 16. Unicast guardians theory.
To conclude the presentation of this example we have to examine `mob_atdl_multicast_refModel` and `mob_atdl_unicast_refModel`. In general, the theory of a reference model includes all the theories developed to realize a particular refinement and provides the designer with all the properties that are satisfied by the refinement, both directly as axioms and as theorems derived from the axioms. In this case, theory `mob_atdl_multicast_refModel` includes `mob_atdl_multicast_G` without adding anything. Something similar can be said about `mob_atdl_unicast_refModel` that simply collects all the properties stated as axioms in the two theories for coordination rules and agents’ behaviour in unicast communications. These two reference model theories have been defined for the sake of uniformity and to follow the general pattern that prescribes to define a support theory and a reference model theory as the ending points of any refinement.

6.1.2 Explicit naming

The most basic type of identification is the one that calls for an explicit naming of the entity we want to refer to and that is based on a unique assignment of names to entities. The profile resolution mechanism provides a generalization to this way of identifying entities for both communication and mobility, but it is possible to use this mechanism to recover the capability of sending messages to one entity or choosing targets of a movement based on their name. Indeed, having a name corresponds to satisfying a property that can be used to build a profile. The key elements of this refinement are the introduction of a profile constructor and the definition of the management of explicit naming based messages or movement requests. Figure 6 shows the MaRK theories for this example and their relationships.

![Figure 6. Theories hierarchy for the explicit naming example.](image)

Following the structure defined in the previous example, we create a support theory, reported in Table 17, in which we define the constructor `name` that uses a component name to build a profile item specifying the identity of an agent or a guardian.
Theory `mob_adtl_prof_name_G`, reported in Table 18, contains the axioms that describe how guardians deal with communication or movement requests based on explicit naming. In `G_name1` a meaning is given to the profiles built using `name`, i.e. any entity satisfies the profile specifying its own name. The next three axioms describe how mobility requests are handled. The idea is that, given axiom `G_name1`, there is no need of looking for a guardian able to resolve the specified profile. Therefore, the guardian of an agent that wants to leave directly deals with the request, `Gml_name` and `Gml’_name`, and forwards it to the guardian of the target neighbourhood, `Gm2_name`. Vetoes can arise as usual. The last two axioms describe how guardians in the case of a communication request manage explicit naming. Once again, the resolution process is started and finished in the first guardian that receives the message.
The theory of the reference model for this example does not simply contains the properties expressed as axioms in the other theories, it provides the designer with a refinement of the general properties for communication and mobility that come from the Mobadtl reference model. Properties M_name and C_name guarantee that the behaviour specified in theory mob_adtl_prof_name_G is actually the expected one: a message delivery request and a movement request where the recipient and the target are identified using their names end up leading to the delivery of the message to the declared recipient and the movement of the agent to the desired neighborhood, if no veto is raised.

- M_name:
  ALL A T H O . EX D .
  l(A, delta(moveTo(consP(name(T),nilP),H))&guardedby(O))
  LEADSTO
  l(A, delta(guardedby(T)) |
    delta(exc(mob(A,O,consP(name(T),nilP)),D)))

- C_name:
  ALL S M R G . EX D .
  l(S, delta(out(M,consP(name(R),nilP)))&guardedby(G))
  LEADSTO
  l(R, delta(in(M,S)))|
  l(S, delta(exc(msg(M,S,consP(name(R),nilP)),D)))

The proofs of these properties are stored in the file mob_adtl_prof_name.ML and can be found in Appendix B. The structure of these proofs is the same as the one for the corresponding properties in the Mobadtl reference model. The difference shows when the transitivity process reaches the points where the axioms defined in mob_adtl_prof_name_G can be used instead of the one from mob_adtl_G1.
6.2 An abstract interface to service discovery

This example aims at showing how Mob\textsubscript{adtl}, intended both as a model and as a methodology, can be used to integrate existing technologies. Specifically, we develop an abstract interface to service discovery mechanisms, but the same reasoning and technique can be adapted to deal with other refinement tasks. The idea is to identify the most general properties that a given technology, routing solution, security policy, etc. provide to agents and guardians and build a reference model that exhibit these properties. Once this reference model is realized, it is possible to base a system specification on it. Any particular implementation choice related to the enacting of the properties is delegated to a further refinement of the reference model.

Service discovery mechanisms are a relevant part in the life of a distributed system and it would be a great benefit to be able to accommodate in the same framework both the description of these mechanisms and the specification of distributed systems to ease the integrations of the two parts. Another appealing feature would be the specification of a general service discovery mechanism, to avoid premature choices with respect to service implementations. This would give the designers the chance to keep different concerns separate, namely the specific system architecture and functionalities on the one side and the service discovery mechanisms on the other. Of course, when the time for a choice has come, it should be possible to detail the general description to match a particular mechanism and still keep the general description as a valid interface between the system and the service providers. Our claim is that Mob\textsubscript{adtl} is such a framework, i.e. Mob\textsubscript{adtl} offers all we need to specify the exploitation of service discovery mechanisms in the most general setting of mobile distributed systems running on WANs, and meets the requirements elicited so far.

We refine the Mob\textsubscript{adtl} model to describe a general abstract interface to service discovery mechanisms, thus actually defining a new reference model that is to be considered as the starting point of the specification of any mobile distributed system that is somehow related to services. This new reference model, together with the existing methodology, is the common framework we are looking for. Then, via a proper refinement process, we can describe any particular service discovery mechanism keeping as the interface the abstract description we gave at the beginning. The compositional nature of Mob\textsubscript{adtl} specifications allows us to think of a scenario like the one represented in Figure 7. Given the specification of a service based system, defined in terms of the Mob\textsubscript{adtl} reference model, we can derive an instantiation of the system depending on the particular discovery mechanism we choose to pick up from a library of mechanism specifications.
The refinement process that leads to the definition of the abstract interface of theory mob_adtl_services_refModel is reflected in the structure presented in Figure 8. At first, we define, as a refinement of the reference model obtained for the name profile, the general properties that a service discovery mechanism must satisfy and that our interface will be called to provide to designers; this is level1 and comprises mob_adtl_services_A1, mob_adtl_services_C1 and mob_adtl_services_G1. Then, we further refine this level to detail the coordination rules and the guardians’ behaviour that guarantee to meet the desired properties; this is level2, and comprises theories mob_adtl_services_C2 and mob_adtl_services_G2. The refinement relation between level 1 and level 2, highlighted in the figure by the dashed lines, follows the general schema from Section 4.4 and guarantees that the reference model exhibits the properties stated at level 1.
how a system can be built starting not directly from the Moladl reference model, but from a refinement that introduces some needed additional property.

The key element of this refinement is the exploitation of the profile resolution mechanism to accommodate the service discovery process. Agents base service requests on a proper profile that specifies both functional and non-functional requirements for the desired service; a guardian will receive the request and the resolution of the profile will lead to the identification of the provider, if any, of the requested service. Theory mob_adtl_services_support, reported in Table 19, contains all the constructors for the predicates and the profiles that are involved in this example.

```
(* everything is needed by services theories *)

mob_adtl_services_support = mob_adtl_prof_name_support +

types
  srv_functional_spec
  srv_params

arities
  srv_functional_spec :: term
  srv_params :: term

consts
  service_available :: [component_name,srv_functional_spec,srv_params] => o
  no_service_available :: [srv_functional_spec,srv_params] => o
  service_req :: [srv_functional_spec,srv_params] => o
  service_profile :: [srv_functional_spec,srv_params,component_name] => profile_item
  ser :: [srv_functional_spec,srv_params] => message
  service_resolve :: [srv_functional_spec,srv_params,component_name] => o
  service_found :: [srv_functional_spec,srv_params,component_name,component_name] => o
  no_service_found :: [srv_functional_spec,srv_params,component_name] => o
  unsatisfiable :: profile => o

End
```

**Table 19. Services support theory.**

Types `srv_functional_spec` and `srv_params` are the types of terms used to specify the functional and non-functional parameter of a service respectively.

Predicates `service_available` and `no_service_available` model the knowledge agents have about the availability of a service and its provider. The first predicate holds in the state of an agent for an entity (agent or guardian) and a service specification if that entity provides to the agent the specified service, the second holds for a service specification if the specified service is not available for the agent. To gain knowledge about the existence of a service provider for a particular service, agents issue a service request modelled by `service_req`. Constructor `service_profile`, which collects the service specification and the name of the agent that issued a service request, is used to build the profile the resolution of which models the
service discovery mechanism. The name of the agent is included in the profile to provide a hook for security, authentication or accounting aspects. The agents can send requests to the provider of a service by building messages using `ser`. Guardians are in charge of searching for the provider of a service and `service_resolve` models in their state the request that can lead to the discovery of a provider, modelled by `service_found`, or to the failure represented by `no_service_found`. When the resolution of a profile gives no results, the profile is said to be unsatisfiable.

**Refinement level 1.** The axioms in theory `mob_adtl_services_A1` state the consistency of agents' knowledge about the availability of a service, `Ser1_1`, and rule the behaviour of agents with regard to the request of a service, which can be issued only once the relative provider has been discovered, `Ser1_2`.

```
mob_adtl_services_A1.thy

(* service discovery interface: agents' behaviour *)

mob_adtl_services_A1 = mob_adtl_refModel + mob_adtl_services_support +
rules

Ser1_1 "ALL A S F P .
co_1(A,~(service_available(S,F,P)&no_service_available(F,P)))"
Ser1_2 "ALL A F P S . co_1(A,out(ser(F,P),consP(name(S),nilP))
  -->service_available(S,F,P))"

end
```

Table 20. Services agents theory, level 1.

Almost all the other axioms of this level belong to the coordination theory, since they are the most abstract specification of a process that leads to the discovery of services and that can involve both guardians and agents. Axiom `Ser1_3` models the outcome of a service request: either the requesting component will get to know the identity of a service provider, or it will be informed that there is no suitable provider. Such knowledge can be acquired only if the proper request has been expressed, as stated in axiom `Ser1_3’`. The last two axioms `Ser1_3’’` and `Ser1_3’’’` express the discovery strategy, i.e., guardians have the responsibility of finding a provider that satisfies the profile in the request.
The only axiom in theory `mob_adtl_services_G1` gives meaning to predicate `unsatisfiable`, asserting that when it holds there is not a set of components that satisfies the profile.

Refinement level 2. Level 2 is a definition of the role guardians play in the process of service discovery. The refinement step leads to a detailed description of the guardians’ behaviour and of the coordination rules on which the discovery process is based. With reference to axioms from `mob_adtl_services_C2` and `mob_adtl_services_G2`, that is what follows a service discovery request. When an agent needs to look for a service, a guardian will be informed, `Ser2_1`, and will take the responsibility of identifying the proper service provider, if any, `Ser2_5`. Of course a guardian will look for a service provider only if an agent asks for it, `Ser2_1'` and `Ser2_5'`, and an agent will receive the result of a service request only if a guardian has received that request, `Ser2_2'`, `Ser2_3'` and `Ser2_2_3'`. Once a guardian resolves the service request, it communicates the result to the requesting agent, `Ser2_2` and `Ser2_3`. Axioms `Ser2_6` and `Ser2_7` describe how the general service discovery mechanism is based on the profile resolution capabilities of guardians already introduced in the general Mob_adtl reference model.

To show that the theories defined at level 2 are a correct refinement of those from level 1, we proved the relation of logical implication to hold

---

Table 21. Services coordination theory, level 1.

Table 22. Services guardians theory, level 1.
between the two levels. This means that we proved all the properties stated in theory mob_adtl_services_C1 to be theorems that can be derived from the axioms of level 2. These proofs are collected in the reference model theory for this example and can be found in Appendix B.

**mob_adtl_services_C2.thy**

(* service discovery refinement: coordination rules *)

mob_adtl_services_C2 = mob_adtl_refModel + mob_adtl_services_support + rules

Ser2_1 "ALL A F P S A .
  1(A,service_reg(F,P)) & guardedby(G) LEADSTO
  1(G,service_resolve(F,P,A))"
Ser2_2 "ALL G F P S A .
  1(G,service_found(F,P,S,A)) LEADSTO
  1(A,service_available(S,F,P))"
Ser2_3 "ALL G F P A .
  1(G,no_service_found(F,P,A)) LEADSTO
  1(A,no_service_available(F,P))"
Ser2_1' "ALL G F P A .
  1(G,service_resolve(F,P,A)) BECAUSE
  1(A,service_reg(F,P)) & guardedby(G)"
Ser2_2' "ALL A S F P . EX G .
  1(A,service_available(S,F,P)) BECAUSE
  1(G,service_found(F,P,S,A))"
Ser2_3' "ALL A F P . EX G .
  1(A,no_service_available(F,P)) BECAUSE
  1(G,no_service_found(F,P,A))"
Ser2_2_3' "ALL A S F P . EX G .
  1(A,service_available(S,F,P))
  no_service_available(F,P)) BECAUSE
  1(G,service_resolve(F,P,A)))"

end

**Table 23. Services coordination theory, level 2.**

**mob_adtl_services_G2.thy**

(* service discovery interface: guardians' behaviour *)

mob_adtl_services_G2 = mob_adtl_services_G1 + rules

Ser2_5 "ALL G F P S A . EX S .
  1(G,service_resolve(F,P,A)) LEADSTO
  1(G,service_found(F,P,S,A) | no_service_found(F,P,A))"
Ser2_6 "ALL G F P S A .
  co_1(G,service_found(F,P,S,A)) -->
  satisfy(consCNL(S,nilCNL),consP(service_profile(F,P,A),nilP)))"
Ser2_7 "ALL F F P A .
  co_1(G,no_service_found(F,P,A)) -->
  unsatisfiable(consP(service_profile(F,P,A),nilP)))"
Ser2_5' "ALL G F P S A .
  1(G,service_found(F,P,S,A) | no_service_found(F,P,A)) BECAUSE
  1(G,service_resolve(F,P,A))"

end

**Table 24. Services guardians theory, level 2.**

Notice that at the reference model level, it is not yet determined how many answers a discovery request will receive, since the number of guardians
that are contacted is also not determined, and left to further refinements. Also the reasons for a negative answer to a service request are not explicitly stated, they are just linked to the profile resolution mechanisms but it is not specified how service profiles are resolved. These are hooks for possible further refinements, one of which could be that a negative answer does not necessarily means that there are no suitable services in the system, but simply that the involved guardians decided not to wait any longer, by some sort of distributed consensus. An example of how to specify such a behaviour using our logic has been specified in [Bib 70] and can be easily exploited in Mobadtl.
6.3 Different kinds of mobility

The rationale of the next example is in the already cited recommendation by Roman, Picco and Murphy from [Bib 1]: find a common framework for the specification and verification of systems based on both physical and logical mobility, join mobile devices and mobile applications, look for fruitful results of this integration. The contribution of the example is therefore twofold. We develop a refinement of the Mob\textsubscript{adtl} to deal with mobile devices, and we exploit this refinement as part of another one dealing with migrating applications, hosted on a network of mobile devices and able to look for a suitable device to move to when the one their are running on has to be disconnected from the net.

This example covers part of the issue highlighted in Section 2.1. In particular, it describes the infrastructure that could support an application like the PC meeting case study. Particular application functionalities can be the subject of further refinements as well as the introduction of a bridge device, with particular routing policies and capabilities to allow the remote participation to the meeting.

6.3.1 Mobile devices

We model a device with a neighbourhood, and refine guardians to deal with physical mobility. The specification of mobile device guardians includes the ability to detect the relevant events, e.g. that another guardian is approaching, to model the awareness that a neighbourhood is reachable. A neighbourhood can be declared reachable or not, depending on further design decisions and technologies adopted. For instance, if Bluetooth technology is adopted, reachability would be a consequence of being in the same network or in a network connected by a bridge. We do not take into account any particular technology, since, as for the service discovery example in Section 6.2, we aim at defining an abstract interface towards physical mobility. Figure 9 shows the MaRK theories developed for this example and their relationships.

![Figure 9. Theories hierarchy for the mobile devices example.](image)

The key elements of this refinement are the predicate constructors that model the means guardians have to interact with their context, i.e. the set of mobile devices they have in their proximity. As for the previous examples, these
Constructors are defined in theory `mob_adtl_mobDevices_support`. Predicates `new` and `exit` model the awareness of the approaching and going away of a device, and the predicate `reachable` models the reachability of a device. Since everything related with identification of entities in Mob_adtl is based on profile resolution, the arrival and the departure of a device is linked to its name and the profile it satisfies, intended as the collection of atomic properties that hold for that device.

```plaintext
 mob_adtl_mobDevices_support.thy
 ("everything is needed by mob_adtl theory for mobile devices ")
 mob_adtl_mobDevices_support = mob_adtl_support +
 const
 new :: [component_name,profile] => o
 exit :: [component_name,profile] => o
 reachable :: component_name => o
 end
```

**Table 25. Mobile devices support theory.**

Looking for a place where to put all the machinery related to the management of ad-hoc networks, and, in particular, the interface between the network environment and the application level, the choice falls naturally on the guardians, that is why this refinement is completely centred on the axioms in theory `mob_adtl_mobDevices_G`.

```plaintext
 mob_adtl_mobDevices_G.thy
 (* specification of the first level of the PC meeting case study :
 guardians' behaviour *)
 mob_adtl_mobDevices_G = mob_adtl_refModel + mob_adtl_mobDevices_support +
 rules
 Gm1 "ALL G A O P T .
 1(G,moveReq(A,O,P,dest(T))) BECAUSE
 1(G,moving(A))&satisfy(consCNL(T,nilCNL),P)&reachable(T))"
 Gm2 "ALL G A O P D .
 1(G,aboutToBeVetoed(mob(A,O,P),D))--guarding(A))"
 Gm3 "ALL A O P D T1 T2 .
 1(G,co1(G,aboutToBeVetoed(mob(A,O,P),D))&(D->dr) &
 reachable(T1)&satisfy(consCNL(T2,nilCNL),P)-->(T1~T2))"
 SR1 "ALL G T P .
 1(G,delta(new(T,P))) LEADSTOC
 1(G,reachble(T))&satisfies(T,P))"
 SR2 "ALL G T P .
 1(G,delta(exit(T,P))) LEADSTOC
 1(G,reachble(T))&satisfies(T,P))"
 SR3 "ALL G T P .
 UNLESS 1(G,new(T,P)),1(G,exit(T,P))"
 end
```

**Table 26. Mobile devices guardians theory.**

Due to the particular setting of ad hoc networks, every distributed form of agreement could be unfeasible or could lead to complex algorithmic solutions that are not in the aim of this example. Therefore, we decided to realize the resolution process of a mobility request in the origin guardian. This means in particular that the origin guardian makes the choice of the destination
neighbourhood based on its knowledge of reachable neighbourhoods, \( G_{\text{md1}} \). Moreover, only the origin guardian can generate mobility vetoes, \( G_{\text{md2}} \), that have to be motivated by the lack of a reachable neighbourhood satisfying the requested profile, \( G_{\text{md3}} \). Guardians are also in charge of reacting to the entrance and leave of neighbourhoods in the net and of keeping their knowledge on reachability and profile satisfaction up to date, \( SR_1 \), \( SR_2 \) and \( SR_3 \).

The reference model theory for this example provides the designer with all the properties described by the presented axioms and can be used as a starting point for the description of any system based on physical mobility.

### 6.3.2 Migrating agents

In our reference scenario, the members of a meeting convene, each one carrying a portable device, like a PDA, a portable computer, etc. We assume that these devices are equipped with wireless technology, and that during the meeting, an ad hoc network is built. Members can join or leave the meeting dynamically, either because they are late or have early planes, or their batteries go down and there are not enough power supplies available in the room. In this scenario, we want to describe applications that can migrate from device to device in order to survive to the disconnection of their computational environment from the network. We exploit the refinement that defines a reference model for mobile devices and take advantage of the capability of guardians of detecting entering and leaving mobile devices. On disconnection, the guardian of a disconnecting device looks for a device apt to host its agents and makes the agents to move towards that device. Figure 10 shows the theories resulting from this refinement and their relationships.

![Diagram](image)

**Figure 10. Theories hierarchy for the migrating agents example.**

Once again, as for the service discovery example of Section 6.2 and for the mobile devices example of Section 6.3.1, what we are looking for is a new reference model, and not a particular system, therefore, the precise definition of the constraints on the migrations is left to a detailed description of the application and of the infrastructure, where a profile can describe the needs
of the application, in terms of computational power, operating system, identity of the owner of the target device, etc.

The support theory of this example introduces the constructors needed to model all the aspects of migrating agents with regard to both the behaviour of agents and of their guardians. Guardians keep trace of the reachable devices that satisfy the meeting profile. The user’s decision to leave the meeting is reflected in the guardian of the user’s device by the event wantToLeave. The guardian looks for a device $T$ that can host its agents and predicate locked$(T)$ denotes that $T$ is the selected guardian. Dually, predicate leaving$(G)$ models, in the guardian $T$ of the receiving device, the fact that it has been selected by $G$. Predicates populationEmpty and moved are used to constrain the leaving of a guardian to the fact that all its agents have completed the migration.

```plaintext
(* everything is needed by the PC meeting case study *)
mob_adtl_migAgents_support = mob_adtl_support +
consts
  meeting :: profile_item
  locked :: component_name => o
  wantToLeave :: o
  leaving :: component_name => o
  populationEmpty :: o
  moved :: component_name => o
end
```

Table 27. Migrating agents support theory.

For the sake of simplicity, we restrict mobility in this reference model to be the consequence of a migration request. With regard to this, axiom Amig from theory mob_adtl_migAgents_A, states that the only mobility request an agent can issue is the one that explicitly address its destination guardian by name; axioms Cmig1 and Cmig1’ from the coordination theory link this kind of requests to the fact that the agent belongs to a leaving neighbourhood and its guardian has chosen a destination guardian to host its agents. Moreover, we assume that there will be not the conditions for double requests, Gmig5, and that guardians do not raise vetoes for mobility requests, Gmig9.

```plaintext
(* specification of the second level of the PC meeting case study : agents' behaviour *)
mob_adtl_migAgents_A = mob_adtl_prof_name_refModel +
mob_adtl_mobDevices_refModel +
mob_adtl_migAgents_support +
rules
  Amig "ALL A P H. EX T . co_l(A,moveTo(P,H)--->P=consP(name(T),nilP))"
end
```

Table 28. Migrating agents agents theory.
In order to make things a bit easier in the orchestration of disconnections and migrations, some assumptions on the behaviour of the participants to the meeting are needed. Theory mob_adtl_migAgents_G formalizes these assumptions. Axioms Asm1 and Asm2 guarantee that the participants leave the meeting if there is still a meeting, i.e. if there are still some devices in the network, and only if no other participant has chosen their devices as migration destinations. Axioms Asm2 also implies that a guardian cannot choose itself as migration destination. Moreover, axiom Gmig6 says that the actual disconnection of a device occurs only when its population is empty. Axioms Gpo1, Gpo2 and Cpo shows how information about the population is gathered. These are examples of safety properties that will not be included into an implementation since they must be guaranteed by the users and not by the application, anyway, without them, nothing interesting can be said and we think it important to include them in the specification.

Another assumption is made about the context, i.e. we assume that guardians are always able to find a proper destination guardian for their agents among the the meeting attendants, Gmig1, Gmig7 and Gmig8.

Axioms Gmig2, Gmig3 and Gmig4 describe how guardians manage the information about the connection state of other guardians, and axioms Gmig1' and Gmig1'' regulate the interplay of locked and leaving.

| Table 29. Migrating agents coordination theory.
The reference model of this example presents some derived properties that has been proved using the axioms in the presented theories, see Appendix B.

- **migAgents_lemma1:**
  \[
  \text{ALL \ G A . EX \ T H .} \\
  \text{l(G,delta(wantToLeave)&guarding(A)) \ LEADSTO} \\
  \text{l(G,satisfy(conS(NL(T,nilCNL),consP(meeting,nilP))))} \\
  \text{&} \\
  \text{l(A,delta(moveTo(consP(name(T),nilP),H))) & guardedby(G))}
  \]

- **migAgents_lemma2:**
  \[
  \text{ALL \ G A . EX \ T OH .} \\
  \text{l(G,delta(wantToLeave)&guarding(A)) \ LEADSTO} \\
  \text{l(G,satisfy(conS(NL(T,nilCNL),consP(meeting,nilP))))} \\
  \text{&} \\
  \text{l(A,delta(guardedby(T)))} \\
  \text{&} \\
  \text{delta(exc(mob(A,O,consP(name(T),nilP)),D)))}
  \]

- **migAgents_lemma3:**
  \[
  \text{ALL \ A OH T D .} \\
  \text{co_l(A,~exc(mob(A,O,consP(name(T),nilP)),D))}
  \]

- **migAgents_lemma4:**
  \[
  \text{ALL \ A OH T D .} \\
  \text{co_l(A,~delta(exc(mob(A,O,consP(name(T),nilP)),D)))}
  \]

- **migAgents_prop:**
  \[
  \text{ALL \ G A . EX \ T .} \\
  \text{l(G,delta(wantToLeave)&guarding(A)) \ LEADSTO}
  \]
l(G, satisfy(consCNL(T, nilCNL), consP(meeting, nilP))) &
   l(A, delta(guardedBy(T)))

These properties show that the coordination pattern designed in the
previously presented theories works well. First, we proved that following a
disconnection announcement, the guardian of the disconnecting device
communicates to all its agents where to migrate, i.e. towards a guardian that
is still in the network, migAgents_lemmal. Following this, the agents will
actually move towards the selected neighbourhood or will receive a veto for
their mobility request, migAgents_lemma2. On the other hand, the axioms
allow proving that the agents do not receive any veto for their mobility
request, migAgents_lemma3 and migAgents_lemma4, and this provides
us with the core property of this refinement, i.e. that all the application
hosted on a disconnecting device will find a proper destination where to
continue their tasks.
7 Explorations on the implementation of the model

Mob_{adtl} model is meant to be used to describe reasonably every mobile system, it is supposed to be a general conceptual tool in the hands of system designers that can choose their preferred programming tools to implement the systems they specified and certified using Mob_{adtl} methodology; nonetheless, having a target programming framework that faithfully reflects the structure of the model would help filling in the gap between specification and implementation and would avoid part of the overhead often associated to the use of formal methods. A process of (semi)automatic translation between specification and implementation would be of great help as well. Moreover, a smooth passage from a formal specification and a working implementation of a system is perfectly in line with the ideas at the base of Mob_{adtl} methodology. For this reason we conducted some exploratory work in the direction of a programming framework based on the Mob_{adtl} model.

When looking for a programming language to implement the model, the choice fell on KLAIM. This choice has been suggested by the similarities between its underlying model and the Mob_{adtl} model: the spawning of new processes seems to be a good mechanism to implement the strong mobility fostered by Mob_{adtl}; the mechanisms for the resolution of logical localities in each node seem to be in line with our ideas about communications that exploit logical names and more in general profiles. Moreover some evolutions of KLAIM well adapt to describe the process of mobility as described in Mob_{adtl}.

The version of KLAIM we chose for our study is the one presented in Section 2.2.3.1.1. Using this reference architecture, Mob_{adtl} model can be mapped in KLAIM as suggested in Figure 11 where Pg and Pc are KLAIM processes that properly implement the semantics of the specifications of a guardian or an agent respectively. Guardians are mapped into nodes that accept connections from other nodes; this allows modeling the relation between an agent and the guardian of its neighborhood: each component is mapped in a KLAIM node that has to ask for a connection to a guardian node in order to be accepted in its neighborhood. Neighborhoods are mapped in the set of agent nodes that are connected to the same guardian node. Such coordination operations are executed by NodeCoordinator processes that are part of Pg and Pc. The connections between guardians are implemented via the connections between the relative guardian nodes.

In this preliminary stage we have done some simplifications on the general model so to be free to focus our attention on the process of “translation” itself rather than on problems like routing policies or security policies. In particular we assumed that:

1. the neighborhood net is a completely connected graph, i.e. each guardian knows, and can interact with, any other guardian;
2. vetos are not raised, any communication or mobility request will be properly served;

The first assumption allowes us to think of a first implementation of the communication process that heavily relies on the mechanisms of logical
localities resolution. Other more realistic solutions that refer to a particular routing would avoid relying on the semantics of KLAiM and would be based on an explicit description of the protocols involved.

Guardians have physical localities as names while components are named using logical localities. When a guardian G is guarding a component C, in the allocation environment of G there is a binding that resolve C in G itself. On the other hand if a guardian is not guarding a component this means that it does not have such binding and the semantics of KLAiM impose it to look for a guardian node that can resolve the component’s name. When an agent wants to send a message to a receiver R, it inserts a tuple of the proper form in the tuple space of its guardian (each component knows it’s guardian name/physical locality). If the guardian is guarding R then it knows its physical locality and can deliver the message inserting the relative tuple in the tuple space of R. If the guardian is not guarding R then it insert the tuple that models the message deliver request in the tuple space located at the physical locality obtained resolving R, i.e. the guardian that is guarding R.

The mobility process is implemented using connection and disconnection operations and creation of new nodes. The processes that implement the guardians and the agents involved in the movements behave accordingly to the axiomatisation of the Mob\_adt reference model as presented in Section 4.4.2. If no vetos are raised the agent can actually move, that is: the process Pc that implements the gent creates a new node and connects it to the guardian node that implements the guardian of the destination neighborhood, then it moves from its original node to this new node and remove the old one as in Figure 12.

Once depicted the architectural correspondences, it is necessary to find a KLAiM translation of properties expressed using ΔSTL(x) operators. The next step is that of depicting a structure of processes Pg and Pc for guardians and components as described in the general model. This structure is meant to be a sort of container that on the one hand implements the general axioms of Mob\_adt, and on the other hand gives the possibility to easily plug the translation of the axioms the describe the peculiarities of a given system. This kind of structure would integrate well with the Mob\_adt methodoloy and would support the refinement process intended as extension of a specification by
adding new axioms that describe already stated properties at lower level of detail or that define new behaviours for agents and guardians.

Figure 12. Agents mobility in KLAIM implementation of Mob_adl
8 Conclusions

The research work behind the development of Mobadtl has been led by the need for a suit of conceptual and technological tools for the development of distributed systems able to cope with the new trends of network technologies, and apt to meet the requirements posed by users that foster a new way of conceiving computing.

The stability assumptions that made classical distributed systems so successful do not hold in mixed networks that integrate a fixed structure with wireless ad hoc networks. Multiple administrative domains, global security policies defined as the composition of many local security policies, the most varied communication mechanisms, mobility of computational nodes and of computing entities, the need applications have to know their current location on the one hand and the chance of not being forced to rely on this knowledge on the other hand, are among the new challenges that define the design and development of new distributed systems.

Looking at the works of many researchers in the area of mobile distributed systems, and building on the lessons learned from software engineering, we depicted the features that characterize Mobadtl, our proposal for the specification and verification of mobile distributed systems:

- a clean conceptual model that help to factorize and structure mobile systems on a common framework;
- a uniform tratement of both physical and logical mobility;
- a strong commitment to the use of formal methods to specify and verify our systems;
- a logic that allows reasoning on the evolution of the states of a set of components that communicate asynchronously via message passing;
- a design style based on refinement and composition;
- a tool that supports the designer in the task of proving the correctness of the specifications;

While defining our conceptual model, we strongly tried not to commit to particular choices as far as communication and security mechanisms are concerned, we just focused our activity on the architectural aspects of the systems. The result is a model that offers a clear description of the main elements of a generic mobile system, prescribes some basic rules on the coordination patterns among these elements, and takes into the proper consideration aspects like different kinds of mobility, location awareness and transparency, multiple security policies, separation of computation and coordination, and flexibility in communications and mobility. Proper hooks are provided in the model to find an accommodation for any particular choice regarding routing and security.

The commitment to the use of formal methods allowed us to define unambiguously the characteristic of our systems, thus posing the basis for the rigorous verification of the properties they are called to satisfy.

We defined $\Delta$DSTL(x), a multimodal logic that exhibits both spatial and temporal modalities: the former are used to express properties of distributed states in terms of the ones exhibited by single components; the latter are
means of stating liveness and safety conditions for systems evolution. Using
our logic we formalized the model and proved the basic structural properties
of communications and mobility. The result of this formalization is a
reference model that is to be used as the starting point for any specification
task based on Mob_adlt. Any system is seen as a refinement of the reference
model and is specified adding details to the general model to describe the
peculiarities of the system, in terms of functionalities, routing of messages
and mobile agents, and security policies. A proper structure is given to the
specification process so to guide the designer through a chain of refinement
steps that leads from the most general reference model to the description of a
particular system.

To ease the application of our methodology, we enriched our proposal
with the adjunct of MaRK, a tool that provides designers with some support
in proving the properties of their systems. MaRK comes equipped with a set
of tactics and pre-proved theorems that can be enriched at any proof session
to form a library that grows depending on the particular needs and domains
of interest of the designer.

Several examples have been developed to test Mob_adlt that showed to
meet our expectations, indeed, the tasks of specifying different kinds of
communications, integrating different kinds of mobility, defining interfaces
towards existing technologies like service discovery, proved straightforward
and succesfull. Besides, MaRK revealed to be a good companion tool for the
Mob_adlt designer since it eased a lot the task of proving the correctness of the
reference model with respect to the requirements we posed on the model, and
of all the examples we developed. Moreover, MaRK greatly helped to achieve
a definitive confidence on the proofs performed by forcing us to avoid error-
prone by hand arguments. Anyway, a lot of work is still to be done. As far as
the model is regarded, we need to cope with many other case studies and to
perform refinements that reach lower levels of detail. As far as the
mechanical proof support is regarded, even if the tool as it is now is greatly
useful, it is also true that the development of proofs is, in some cases, still too
interactive. For this reason, we need to depict new more powerful tactics that
help to automatize the verification process.

Besides providing a set of conceptual and technological tools to the
designers, our aim is also that of providing implementers with some means to
achieve a seamless translation from specifications to implementations. With
regard to this aim, our reference formal model can be seen as the
specification of a programmative framework on which to base the
implementation of mobile distributed systems whose architecture faithfully
reflects that of our model. Following this idea, we have conducted an initial
exploration on a possible implementation of our model that led to find in
KLAIM a promising language for the implementation of Mob_adlt. This
research direction has bee just opened and many other things still need to be
done among which a a wider study of the available programming languages,
and, of course, the first real implementation experiments.

In spite of what we still have to do, we believe that our results constitute
a good starting point and already represent a convincing achievement in the
search for the aforementioned suit of tools, since, as shown, Mob_adlt can be
successfully used to specify and verify mobile distributed systems.
Bibliography


Bib 23. Francesca Levi and Davide Sangiorgi. Mobile safe ambients. Full version of [Bib 24].


Bib 64. C. Weidenbach, B. Gaede, G. Rock. Spass and Flotter Version 0.42. In [Bib 66], pages 141-145.


Bib 86. Tesi Ciccio.


Appendix A

This appendix contains the collection of all the Isabelle theories that have been used to build Mark, a presented in Section 5.2. The theories are presented in their completeness, as they come from the .thy files in which they are organized. For a collection of all the proofs based on these theories that have been performed see Appendix B.

The logic theories

PC.thy

PC = FOL +

rules

logicLemma1 "A<->B ==> A=B"
logicLemma2 "A=B ==> A<->B"

ContrapositionPN "(F1<->F2) ==> (~F2<->~F1)"
ContrapositionNP "(~F1<->~F2) ==> (F2<->F1)"
NotIffPN "F1<->F2 ==> F1<->F2"
NotIffNP "~F1<->~F2 ==> F1<->F2"
SwapIff "F1<->F2 ==> F2<->F1"

Iff_def "F1<->F2==(F1<->F2) & (F2<->F1)"
IffSimm "A<->A"
IffTrans "(F1<->F3) == (F1<->F2) & (F2<->F3)"
IffI "[|F1<->F2; F2<->F1|] ==> F1<->F2"
IffE_right "F1<->F2 ==> (F1<->F2)"
IffE_left "F1<->F2 ==> (F2<->F1)"

Imp_def1 "~A|B ==> A<->B"
Imp_def2 "A--B == ~A|B"
Imp_def "A<->B == A|B"
Imp_refl "A<->A"
Imp_trans "[|A<->B; B<->C|] ==> A<->C"

demorgan1 "~(A & B) == ~A & ~B"
demorgan2 "~(A | B) == ~A & ~B"

Or_rotate "A|B == B|A"
OrAnd_distr "A & B | C == (A | C) & (B | C)"
AndOr_distr "A & (B | C) == (A & B) | (A & C)"
AndIdemp "A & True == A"

end
DSL.thy

(* DSL: Distributed State Logic *)

DSL = PC +

types

    component_name

arities

    component_name :: term

consts

    l :: [component_name, o] => o
    co_l :: [component_name, o] => o

defs

    co_l_def "co_l(M,F) == ~l(M,~F)"
    l_def "l(M,F) == ~co_l(M,~F)"

rules

(* DSL axioms *)

    K "co_l(M,F1---F2)---(co_l(M,F1)---co_l(M,F2))"
    DSl1 "co_l(M,F<->co_l(M,F))"
    DSl2 "~eq(M,N)-->co_l(M,co_l(N,False))"
    Nec "F===>co_l(M,F)"

(* some useful rules related to PC and DSL operators *)

    NotNotE_mf1_Iff_f2
    "l(M,~F1)---F2==>
     l(M,F1)---F2"

    NotNotE_mf1_Imp_mf2
    "l(M,~F1)---l(M,~F2)==>
     l(M,F1)---l(M,F2)"

    NotNotE_mf1_Imp_mf2
    "l(M,~F1)---l(M,~l(M,~F2))==>
     l(M,F1)---l(M,F2)"

    NotNotE_mf1_Imp_mf2
    "l(M,~l(M,~F1))---l(M,~F2)==>
     l(M,l(M,F1))---l(M,F2)"

    NotNotE_mf1_Iff_mf2
    "l(M,~F1)<--l(M,~l(M,~F2))==>
     l(M,l(M,F1))<--l(M,F2)"

    NotNotE_mf1_Iff_mf2
    "l(M,~F2)<--l(M,~l(M,~F1))==>
     l(M,F2)<--l(M,l(M,F1))"
NotNotE_mf1_Imp_mf2ANDmf3
"l(M, ~F1) --> (l(M, ~F2) & l(M, ~F3)) ==>
l(M, F1) --> (l(M, F2) & l(M, F3))"

NotNotE_mf1_Imp_mf2ORmf3
"l(M, ~F1) --> (l(M, ~F2) | l(M, ~F3)) ==>
l(M, F1) --> (l(M, F2) | l(M, F3))"

end
DSTL.thy

(* dDSTL: Distributed State Temporal Logic *)
dDSTL = DSL +
consts

(* event operator *)
  delta :: o => o
(* dsl operators *)
  leads_to :: [o,o] => o ("_ LEADSTO _")
  leads_to_c :: [o,o] => o ("_ LEADSTOC _")
  because :: [o,o] => o ("_ BECAUSE _")
  because_c :: [o,o] => o ("_ BECAUSEC _")
  init :: o => o ("INIT")
  stable :: o => o ("STABLE")
  unless :: [o,o] => o ("UNLESS")
defs
  unless_def "STABL(F) == UNLESS(F,False)"
rules

(* rules on events *)

(*ev is idempotent *)
  idemp_ev "delta(delta(A))="delta(A)"

(* relations between conditions and events *)
  cond_eventP "A-->delta(-A)"
  cond_eventN "-A-->delta(A)"

(* introduction and elimination rules *)
  LI "F LEADSTOC F"
  BiC "F BECAUSEC F"
  LI "F1 LEADSTOC F2 ==> F1 LEADSTO F2"
  BI "F1 BECAUSEC F2 ==> F1 BECAUSE F2"
  UI "UNLESS(F,F)"
  InI "F==>INIT(F)"
  SI "F==>STABLE(F)"
  SE "[| INIT(l(M,F));STABLE(l(M,F))|] ==> co_1(M,F)"
  LE "F LEADSTO False ==> ~F"
  BE "[| INIT(-F);F BECAUSE False|] ==> ~F"
  BSE "[| l(M,F) BECAUSE l(M,G);STABLE(l(M,G))|] ==> co_1(M,F-->H)"
  deltaE "delta(F)-->F"
  deltaI "F LEADSTO G ==> (F&~G) LEADSTO delta(G)"

(* transitivity rules *)
  LTR "[| F1 LEADSTO F2;F2 LEADSTO F3|] ==> F1 LEADSTO F3"
  BTR "[| F1 BECAUSE F2;F2 BECAUSE F3|] ==> F1 BECAUSE F3"
  UC "[| UNLESS(l(M,F1),l(M,F2));UNLESS(l(M,F2),l(M,F3))|] ==> 
UNLESS(l(M,F1)\|l(M,F2),l(M,F3))"

(* premises and consequences strengthening and weakening *)

LSW "|G1--->F1;F1 LEADSTO F2;F2--->G2|=>G1 LEADSTO G2"
LPD "|F1 LEADSTO G;F2 LEADSTO G|=>(F1|F2) LEADSTO G"
LCC "|[G LEADSTO F1;G LEADSTO F2|=> G LEADSTO (F1&F2)"
BSW "|G1--->F1;F1 BECAUSE F2;F2--->G2|=> G1 BECAUSE G2"
BPD "|F1 BECAUSE G;F2 BECAUSE G|=>(F1|F2) BECAUSE G"
BCC "|because(G,F1);because(G,F2)|=>because(G,F1&F2)"
LcSW "|[G1--->F1;F1 LEADSTO F2);F2--->G2]|=>
G1 LEADSTOC G2"
LcPD "|[F1 LEADSTOC G);F2 LEADSTOC G|=>
(F1|F2) LEADSTOC G"
LcCC "|[G LEADSTOC F1;G LEADSTOC F2|=>
G LEADSTOC (F1&F2)"
BcSW "|[G1--->F1;F1 BECAUSESEC F2;F2--->G2]|=>G1 BECAUSESEC G2"
BcPD "|[F1 BECAUSESEC G;F2 BECAUSESEC G|=>(F1|F2) BECAUSESEC G"
BcCC "|[G BECAUSESEC F1;G BECAUSESEC F2|=>G BECAUSESEC (F1&F2)"

(* notification *)

Notif "|[F BECAUSE G1;G1 LEADSTO l(M,G2);STABLE(l(M,G2))|]
=>(F&l(M,True)) LEADSTO l(M,G2)"

(* confluence *)

Conf "|[STABLE(l(M,F1));STABLE(l(M,F2))|]|=>
(l(M,F1)&l(M,F2))--->l(M,F1\&F2)"

(* properties of the initial state *)

I1 "INIT(l(M,True))"
I2 "INIT(l(M,F))=>INIT(co_l(M,F))"
I3 "INIT(co l(M,F))=>INIT(l(M,F))"
IW "|[INIT(F);F--->G]|=>INIT(G)"
IC "|[INIT(F);INIT(F1)]|=>INIT(F\&F1)"

(* useful theorems *)

(* from a couple of unless to stable *)
U2S "|[UNLESS(l(M,F1),l(M,F2));UNLESS(l(M,F2),l(M,F1))|]|=>
STABLE(l(M,F1\&F2))"

SW "|[STABLE(F);F--->G]|=>STABLE(G)"

(* knowledge of the past and stability assumptions give
knowledge on the future *)

LTB1 "|[l(M,F1) LEADSTO l(N,F2);l(M,F1) BECAUSE l(N,F3);
l(N,F3)--->l(N,F4);STABLE(l(N,F4))|]|=>
l(M,F1) LEADSTO l(N,F2\&F4)"

(* injecting true conditions in the premises of a leads_to *)
LTtrueCond "|[l(A,F1\&F3) LEADSTO F2;co_l(A,F3)]|=>
l(A,F1) LEADSTO F2"

(* disjunction of because consequences *)
BOR "|[F1 BECAUSE (F4|F3);F4 BECAUSE F2]|=>
F1 BECAUSE (F2|F3)"

151
(*) a theorem for conjunction of because consequences *)
BAand "[(F1 BECAUSE F2; F1 BECAUSE F3)] ==> F1 BECAUSE (F2 & F3)"

(*) disjunction elimination and consequences of because and leads to *)
BdisjE "[(F BECAUSE (P | Q); F BECAUSE P ==> R;
           F BECAUSE Q ==> R)] ==> R"
LdisjE "[(F LEADSTO (P | Q); F LEADSTO P ==> R;
          F LEADSTO Q ==> R)] ==> R"

(*) negative invariants introduction *)
co_1 notI "[(l(M,F1) BECAUSE l(N,F2); co_1(N, ~F2);
              INIT(l(M, ~F1)))] ==> co_1(M, ~F1)"

end
The model theories

mob_adtl_support.thy

(* everything is needed by mob_adtl models: types, structures, predicates... *)

mob_adtl_support = dDSTL +

types

component_name_list (* list of component names *)

profile_item
profile (* the type of terms used to model profiles in mobility and communications requests *)

message (* the type for messages *)
req_state (* the type for modelling the state of mobility and communication requests *)

exc_type (* the type for exceptions *)
exc_data (* the type for the data relative to exceptions *)

arities

component_name_list :: term
profile_item :: term
profile :: term
message :: term
req_state :: term
exc_type :: term
exc_data :: term

consts

(* constructors for component names lists *)
nilCNL :: component_name_list
consCNL :: [component_name,component_name_list] => component_name_list

(* operators for component names lists *)
(* first list is sublist of second list *)
subCNL :: [component_name_list,component_name_list] => o
(* component name is in list *)
isInCNL :: [component_name,component_name_list] => o
(* equality of two lists *)
eqCNL :: [component_name_list,component_name_list] => o

(* constructors for profiles *)
nilP :: profile
consP :: [profile_item,profile] => profile
(* operators for profiles *)
impliesP :: [profile,profile] => o
isInP :: [profile_item,profile] => o

(* equivalence of req_state and exc_data *)
req_state_eq :: [req_state,req_state] => o
exc_data_eq :: [exc_data,exc_data] => o
(* standard predicates from the requirements specifications
   and the reference model *)
guardedBy :: component_name => o
moveTo :: [profile, component_name_list] => o
history :: component_name_list => o
satisfy :: [component_name_list, profile] => o
satisfies :: [component_name, profile] => o
satisfies_constraint :: [component_name, profile_item] => o
guarding :: component_name => o
moving :: component_name => o
mobReq :: [component_name, component_name, profile, req_state] => o

msgReq :: [message, component_name, profile, req_state] => o
msg :: [message, component_name, profile] => exc_type
mob :: [component_name, component_name, profile] => exc_type
exc :: [exc_type, exc_data] => o
toBeVetoed :: [exc_type, exc_data] => o
in :: [message, component_name] => o
out :: [message, profile] => o
i :: req_state
u :: req_state
dest :: component_name => req_state
rec :: component_name => req_state
dr :: exc_data
defs

eqCNL_def "eqCNL(L1,L2) == subCNL(L1,L2) & subCNL(L2,L1)"

rules

(* definition of sublists *)
isInCNL_rule "[|((C1=C2)|isInCNL(C1,L))|] =>
   isInCNL(C1,consCNL(C2,L))"
subCNL_nil "subCNL(nilCNL,L)"
subCNL_rule "[(isInCNL(C,L2);subCNL(L1,L2))|] =>
   subCNL(consCNL(C,L1),L2)"
eqCNL_rule "~subCNL(L2, consCNL(C, L2))"

end
mob_adtl_A0.thy

(* mob_adtl requirements specification: theory A0, agents' behaviour *)

mob_adtl_A0 = mob_adtl_support +

rules

U "ALL A G1 G2 .
   co _l(A, guardedby(G1)&guardedby(G2)--eq(G1,G2))"
S1 "ALL A P H . _l(A, moveTo(P,H)--history(H))"
S2 "ALL A G1 H . EX G2 .
   UNLESS(_l(A,guardedby(G1)&history(H)),
       _l(A,guardedby(G2)&history(consCNL(G1,H))))"
S3 "ALL A G H . EX P .
   _l(A,history(consCNL(G,H))) BECAUSE _l(A,moveTo(P,H))"
S4 "ALL A . init(_l(A,history(nilCNL)))"
S5 "ALL A . co _l(A, EX G H . guardedby(G) & history(H))"

end
(* mob_adtl requirements specification: theory C0, coordination rules *)

mob_adtl_C0 = mob_adtl_support +

rules

M "ALL A P H O . EX T D .
  l(A,delta(moveTo(P,H)))&guardedby(O)) LEADSTO
  l(A,delta(exc(mob(A,O,P),D)))"

M1 "ALL A T . EX F H G1 O St G2 D.
  l(A,guardedby(T)) BECAUSE
  l(A,delta(moveTo(F,H)))&
  l(G1,mobReq(A,O,P,St)&satisfy(consCNL(T,nilCNL),P))| l(G2,toBeVetoed(mob(A,T,P),D)))"

A1 "ALL A T . EX O P St1 St2 G D1 .
  l(A,guardedby(T)) BECAUSE
  l(T,mobReq(A,O,P,St1))&
  l(O,mobReq(A,O,P,St2))| l(G,toBeVetoed(mob(A,T,P),D1)))";

M2 "ALL A O P D . EX G St H .
  l(A,exc(mob(A,O,P),D)) BECAUSE
  l(A,delta(moveTo(F,H)))&
  l(G,mobReq(A,O,P,St))& l(G,toBeVetoed(mob(A,O,P),D)))"

Id1 "ALL G A O P St . EX H .
  l(G,mobReq(A,O,P,St)) BECAUSE l(A,delta(moveTo(P,H)))";

C "ALL S M P G . EX R D .
  l(S,guardedby(G)) LEADSTO
  l(R,delta(in(M,S))))|l(S,delta(exc(msg(M,S,P),D)))";

C1 "ALL R M S . EX P G St .
  l(R,exc(M,S)) BECAUSE
  l(S,guardedby(G))&
  l(S,delta(out(M,P)))& l(G,guarding(S))& l(G,delta(msgReq(M,S,P,St))&satisfy(consCNL(R,nilCNL),P)))"

A2 "ALL R M S . EX G1 P St1 G2 St2 .
  l(R,guarding(R))&
  l(G1,guarding(S)))& l(G2,guarding(S)))"

C2 "ALL S M P D . EX G St .
  l(S,guarding(S)) BECAUSE
  l(S,guarding(S))&
  l(S,guarding(S))& l(G,guarding(S))&
  l(G,delta(msgReq(M,S,P),D)))"

Id2 "ALL G M S P St .
  l(G,delta(msgReq(M,S,P,St))) BECAUSE
  l(S,guarding(S))"

end
mob_adtl_G0.thy

(* mob_adtl requirements specification: theory G0, guardians' behaviour *)

mob_adtl_G0 = mob_adtl_support +

rules

N "ALL G1 G2 . co_l(G1,~guardedby(G2))"
Sat1 "ALL G S P C . co_l(G,satisfy(S,P)--> (isInCNLS(C,S)-->satisfies(C,P)))"
Sat2 "ALL G S P Pi . co_l(G,satisfies(S,P)--> (isInP(Pi,P)--> >satisfies_constraint(S,Pi)))" end

mob_adtl_A1.thy

(* mob_adtl reference model: theory A1, components' behaviour *)

mob_adtl_A1 = mob_adtl_A0 +

rules

Ia "ALL A P1 H M P2 Type D .
   INIT(l(A,~moveTo(P1,H)&~out(M,P2)&~exc(Type,D)))" end
mob_adtl_C1.thy

(* mob_adtl reference model: theory C1, coordination rules *)
mob_adtl_C1 = mob_adtl_support +

rules

ILA "ALL A . EX G . init(l(A, guardedby(G)) & l(G, guarding(A)))"

Cm1 "ALL APHO .
    l(A, delta(moveTo(P, H)) & guardedby(O)) LEADSTO
    l(O, delta(mobReq(A, O, P, i)))"

Cm2 "ALL A T .
    l(T, delta(guarding(A))) LEADSTO l(A, delta(guardedby(T)))"

Cm3 "ALL G A O P D .
    l(G, delta(toBeVetoed(mob(A, O, P), D))) LEADSTO
    l(A, delta(exc(mob(A, O, P), D)))"

Cm1' "ALL O A P . EX H .
    l(O, mobReq(A, O, P, i)) BECAUSE
    l(A, delta(moveTo(P, H)) & guardedby(O))"

Cm2' "ALL A T . l(A, guardedby(T)) BECAUSE l(T, guarding(A))"

Cm3' "ALL A O P D . EX G .
    l(A, exc(mob(A, O, P), D)) BECAUSE
    l(G, toBeVetoed(mob(A, O, P), D))"

Cc1 "ALL S M P G .
    l(S, delta(out(M, P)) & guardedby(G)) LEADSTO
    l(G, delta(msgReq(M, S, P, u)))"

Cc2 "ALL G M S P R . EX D .
    l(G, delta(msgReq(M, S, P, rec(R))) & guarding(R)) LEADSTOC
    l(R, delta(in(M, S))) | l(G, delta(toBeVetoed(msg(M, S, P), D)))"

Cc3 "ALL G M S D .
    l(G, delta(toBeVetoed(msg(M, S, P), D))) LEADSTO
    l(S, delta(exc(msg(M, S, P), D)))"

Cc1_1' "ALL G M S P . EX G1 .
    l(G1, delta(msgReq(M, S, P, u))) BECAUSE
    l(G1, delta(msgReq(M, S, P, u)) & guardedby(S)))"

Cc1_2' "ALL G M S P .
    l(G, delta(msgReq(M, S, P, u)) & guardedby(S)) BECAUSE
    l(S, delta(out(M, P)) & guardedby(G)))"

Cc2' "ALL R M S . EX G P .
    l(R, in(M, S)) BECAUSE
    l(G, delta(msgReq(M, S, P, rec(R))) & guarding(R))"

Cc3' "ALL S M P D . EX G .
    l(S, exc(msg(M, S, P), D)) BECAUSE
    l(G, toBeVetoed(msg(M, S, P), D))"
A3 "ALL G A. UNLESS(l(G,guarding(A)),l(G,moving(A)))"

A4 "ALL A G. UNLESS(l(G,moving(A)),l(G,guarding(A)))"

A5 "ALL G A. co_l(G,¬(guarding(A) & moving(A)))"

end
mob_adtl_G1.thy

(* mob_adtl reference model: theory G1, guardians' behaviour *)

mob_adtl_G1 = mob_adtl_G0 +

rules

Gm1 "ALL O A P . EX G D.
  l(O,delta(mobReq(A,O,P,i)) & guarding(A)) LEADSTO
  l(G,delta(mobReq(A,O,P,u))) |
  l(O,delta(toBeVetoed(mob(A,O,P),D)))"

Gm2 "ALL G A O P . EX T D.
  l(G,delta(mobReq(A,O,P,u))) LEADSTO
  l(T,delta(mobReq(A,O,P,dest(T)))) |
  l(G,delta(toBeVetoed(mob(A,O,P),D)))"

Gm3 "ALL T A O P . EX D.
  l(T,delta(mobReq(A,O,P,dest(T)))) LEADSTO
  l(T,delta(guarding(A)))) |
  l(T,delta(toBeVetoed(mob(A,O,P),D)))"

Gm4 "ALL O A P .
  l(O,delta(mobReq(A,O,P,i)) & guarding(A)) LEADSTOC
  l(O,delta(moving(A))))"

Gm5 "ALL O A P .
  l(O,delta(mobReq(A,O,P,i)) & moving(A)) LEADSTO
  l(O,delta(toBeVetoed(mob(A,O,P),dr))))"

Gm6 "ALL G A O P D .
  l(G,delta(toBeVetoed(mob(A,O,P),D)) & ~exc_data_eq(D,dr))
LEADSTO l(G,guarding(A)))"

Gm1' "ALL G A O P .
  l(G,mobReq(A,O,P,u)) BECAUSE
  l(O,(mobReq(A,O,P,i) & guarding(A))))"

Gm2' "ALL T A O P . EX G1 .
  l(T,mobReq(A,O,P,dest(T))) BECAUSE
  l(G1,mobReq(A,O,P,u) & satisfy(consCNL(T,nilCNL),P)))"

Gm3_6' "ALL G A . EX O P G1 D .
  l(G,guarding(A)) BECAUSE
  l(G,mobReq(A,O,P,dest(G))) |
  l(G1,toBeVetoed(mob(A,G,P),D)))"

Gm1_2_3' "ALL G A O P D . EX St .
  l(G,toBeVetoed(mob(A,O,P),D)) BECAUSE
  l(G,mobReq(A,O,P,St)))"

Gm4' "ALL O A . EX P .
  l(O,moving(A)) BECAUSE l(O,mobReq(A,O,P,i))"

Gm5' "ALL G A O P .

160
l(G, toBeVetoed(mob(A, G, P), dr)) BECAUSE
l(G, mobReq(A, G, P, i) & moving(A))"

Gm7 "ALL O A P Pl .
co_1(G, (mobReq(A, O, P, i) & mobReq(A, O, Pl, i)) -->
(impliesP(P, Pl) & impliesP(Pl, P)))"

Gm8 "ALL G A O P St D . EX G1 T .
l(G, mobReq(A, O, P, St)) -->
l(G, mobReq(A, O, P, i)) |
l(G, mobReq(A, O, P, u)) |
l(T, mobReq(A, O, P, dest(T)))"

Gm9 "ALL G A O P St D . EX G1 T .
l(G, mobReq(A, O, P, St)) & l(G, toBeVetoed(mob(A, O, P), D)) -->
l(G, mobReq(A, O, P, i)) & l(G, toBeVetoed(mob(A, O, P), D)) |
l(G, mobReq(A, O, P, u)) & l(G, toBeVetoed(mob(A, O, P), D)) |
l(T, mobReq(A, O, P, dest(T))) &
l(T, toBeVetoed(mob(A, O, P), D))"

Gc1 "ALL G M S P . EX G1 R D .
l(G, delta(msgReq(M, S, P, u))) LEADSTO
l(G, delta(msgReq(M, S, P, rec(R)))) |
l(G, delta(toBeVetoed(msg(M, S, P), D)))"

Gc2 "ALL G M S P R . EX G1 .
l(G, delta(msgReq(M, S, P, rec(R))) & guarding(R)) LEADSTO
l(G, delta(msgReq(M, S, P, rec(R))) & guarding(R)))"

Gc1_2' "ALL G M S P R . EX G1 .
l(G, delta(msgReq(M, S, P, rec(R)))) BECAUSE
l(G, delta(msgReq(M, S, P, u)) & satisfy(consCNL(R, nilCNL), P))"

Gc1_2' ' "ALL M S P D . EX St .
l(G, toBeVetoed(msg(M, S, P), D)) BECAUSE
l(G, delta(msgReq(M, S, P, St))))"

Gc3 "ALL G M S P St . EX R .
l(G, delta(msgReq(M, S, P, St))) -->
l(G, delta(msgReq(M, S, P, u)))) |
l(G, delta(msgReq(M, S, P, rec(R))))"

Ig "ALL A O P St D .
INIT(l(G, ~mobReq(A, O, P, St) & msgReq(M, S, P, St) &
~toBeVetoed(mob(A, O, P), D) &
~toBeVetoed(msg(M, S, P), D)))"

end

mob_adtl_refModel.thy

(* mob_adtl reference model *)

mob_adtl_refModel = mob_adtl_A1 + mob_adtl_G1 + mob_adtl_C1 +
end
The theories of the examples

mob_adtl_multicast_support.thy

(* everything is needed by multicast theories *)

mob_adtl_multicast_support = mob_adtl_support +

consts

  multi :: profile_item

end

mob_adtl_multicast_G.thy

(* specification of unicast communications in mob_adtl: guards' behaviour *)

mob_adtl_multicast_G = mob_adtl_refModel +
  mob_adtl_multicast_support +

rules

  Multil "ALL M S P R . EX D .
  l(G,delta(msgReq(M,S,consP(multi,P),u)) &
    satisfy(consCNL(R,nilCNL),consP(multi,P)))" LEADSTO
  l(G,delta(msgReq(M,S,consP(multi,P),rec(R))))|
  l(G,toBeVetoed(msg(M,S,consP(multi,P),D)))"

end

mob_adtl_multicast_refModel.thy

(* mob_adtl reference model for multicast communication *)

mob_adtl_multicast_refModel = mob_adtl_multicast_G +

end
mob_adtl_unicast_support.thy

(* everything is needed by unicast theories *)

mob_adtl_unicast_support = mob_adtl_support +

consts

    uni :: profile_item

end

mob_adtl_unicast_C.thy

(* specification of unicast communications in mob_adtl: coordination rules *)

mob_adtl_unicast_C = mob_adtl_refModel +
    mob_adtl_unicast_support +

rules

    Uni1 "ALL G M S P .
        l(G,delta(msgReq(M,S,consP(uni,P),u))) BECAUSE
        l(S,delta(out(M,consP(uni,P)))&guardedby(G))"

end

mob_adtl_unicast_G.thy

(* specification of unicast communications in mob_adtl: guardians' behaviour *)

mob_adtl_unicast_G = mob_adtl_refModel +
    mob_adtl_unicast_support +

rules

    Uni2 "ALL G M S P R1 R2 .
        l(G,msgReq(M,S,consP(uni,P),rec(R1))&
        msgReq(M,S,consP(uni,P),rec(R2))-->eqCN(R1,R2))"

    Uni3 "ALL G M S P R .
        STABLE(l(G,msgReq(M,S,consP(uni,P),rec(R))))"

end
**mob_adtl_unicast_G.thy**

(* specification of unicast communications in mob_adtl: 
guardians' behaviour *)

mob_adtl_unicast_G = mob_adtl_refModel +
mob_adtl_unicast_support +

rules

Uni2 "ALL G M S P R1 R2 .
    l(G,msgReq(M,S,consP(uni,P),rec(R1)) &
        msgReq(M,S,consP(uni,P),rec(R2)) --\rightarrow eqCN(R1,R2))"

Uni3 "ALL G M S P R .
    STABLE(l(G,msgReq(M,S,consP(uni,P),rec(R)))))"

end

**mob_adtl_unicast_refModel.thy**

(* mob_adtl reference model for unicast communication *)

mob_adtl_unicast_refModel = mob_adtl_unicast_G +
    mob_adtl_unicast_C +

end
mob_adtl_prof_name_support.thy

(* everything is needed by mob_adtl theory for profile name(X) *)
mob_adtl_prof_name_support = mob_adtl_support +

consts
  name :: component_name => profile_item

end

mob_adtl_prof_name_G.thy

(* theory for profile name(X): guardians' behaviour *)
mob_adtl_prof_name_G = mob_adtl_refModel +
  mob_adtl_prof_name_support +

rules
  G_name1 "ALL G X .
    co_l(G,satisfy(consCNL(X,nilCNL),consP(name(X),nilP)))"

  Gm1_name "ALL O A T. EX D .
    l(O,delta(mobReq(A,O,consP(name(T),nilP),i)) &
       guarding(A)) LEADSTO
    l(O,delta(mobReq(A,O,consP(name(T),nilP),u))) | l(O,delta(toBeVetoed(mob(A,O,consP(name(T),nilP),D))))"

  Gm2_name "ALL G A O T . EX D .
    l(G,delta(mobReq(A,O,consP(name(T),nilP),u)))
    LEADSTO
    l(T,delta(mobReq(A,O,consP(name(T),nilP),dest(T))) | l(G,delta(toBeVetoed(mob(A,O,consP(name(T),nilP)),D))))"

  Gm1'_name "ALL G A O P .
    l(O,mobReq(A,O,consP(name(T),nilP),u)) BECAUSE
    l(O, (mobReq(A,O,consP(name(T),nilP),i) & guarding(A)))"

  Gcl_name "ALL G M S R. EX D .
    l(G,delta(msgReq(M,S,consP(name(R),nilP),u)))
    LEADSTO
    l(G,delta(msgReq(M,S,P,rec(R)))) | l(G,delta(toBeVetoed(msg(M,S,P),D)))"

  Gc1_2'_name "ALL G M S P R . EX G1 .
    l(G,delta(msgReq(M,S,P,rec(R)))) BECAUSE
    l(G,delta(msgReq(M,S,consP(name(R),nilP),u)) &
       satisfy(consCNL(R,nilCNL),P))"

end
mob_adtl_prof_name_refModel.thy

(* mob_adtl reference model *)
mob_adtl_refModel = mob_adtl_A1 + mob_adtl_G1 + mob_adtl_C1 +

end
mob_adtl_services_support.thy

(* everything is needed by services theories *)

mob_adtl_services_support = mob_adtl_prof_name_support +

types

  srv_functional_spec
  srv_params

arities

  srv_functional_spec :: term
  srv_params :: term

consts

  service available ::
    [component_name,srv_functional_spec,srv_params] => o
  no_service_available :: [srv_functional_spec,srv_params] => o
  service_req :: [srv_functional_spec,srv_params] => o
  service_profile ::
    [srv_functional_spec,srv_params,component_name] => profile_item
  ser :: [srv_functional_spec,srv_params] => message
  service_resolve ::
    [srv_functional_spec,srv_params,component_name] => o
  service_found ::
    [srv_functional_spec,srv_params,component_name, component_name]=>o
  no_service_found ::
    [srv_functional_spec,srv_params,component_name] => o
  unsatisfiable :: profile => o

end

mob_adtl_services_A1.thy

(* service discovery interface: agents' behaviour *)

mob_adtl_services_A1 = mob_adtl_refModel +
  mob_adtl_services_support +

rules

  Ser1_1  "ALL A S F P .
    co_1(A, ~(service_available(S,F,P) &
      no_service_available(F,P)))"
  Ser1_2  "ALL A F P S .
    co_1(A,out(ser(F,P),consP(name(S),nilP)) -->
      service_available(S,F,P))"

end
**mob_adtl_services_C1.thy**

(* service discovery interface: coordination rules *)

```plaintext
mob_adtl_services_C1 = mob_adtl_refModel +
                     mob_adtl_services_support +

rules

Ser1_3  "ALL A F P G . EX S .
        l(A, delta(service_req(F,P)) & guardedby(G)) LEADSTO
        l(A, service_available(S,F,P) | no_service_available(F,P))"

Ser1_3' "ALL A S F P . EX G .
        l(A, service_available(S,F,P) | no_service_available(F,P))
        BECAUSE l(A, delta(service_req(F,P)) & guardedby(G))"

Ser1_3'' "ALL S F P . EX G .
          l(A, service_available(S,F,P)) BECAUSE
          l(G, satisfy(consCNL(S, nilCNL),
                       consP(service_profile(F,P,A), nilP)))"

Ser1_3''' "ALL A F P . EX G .
          l(A, no_service_available(F,P)) BECAUSE
          l(G, unsatisfiable(consP(service_profile(F,P,A), nilP)))"
```

end

**mob_adtl_services_G1.thy**

(* service discovery interface: guardians' behaviour *)

```plaintext
mob_adtl_services_G1 = mob_adtl_refModel +
                      mob_adtl_services_support +

rules

Ser1_4  "ALL P Set . l(G, unsatisfiable(F) --> ~satisfy(Set,F))"
```

end
mob_adtl_services_C2.thy

(* service discovery refinement: coordination rules *)

mob_adtl_services_C2 = mob_adtl_refModel +
  mob_adtl_services_support +

rules

  Ser2_1  "ALL A F P G .
           l(A,delta(service_req(F,P))&guardedby(G)) LEADSTO
           l(G,service_resolve(F,P,A))"
  Ser2_2  "ALL G F P S A .
           l(G,service_found(F,P,S,A)) LEADSTO
           l(A,service_available(S,F,P))"
  Ser2_3  "ALL G F P A .
           l(G,no_service_found(F,P,A)) LEADSTO
           l(A,no_service_available(F,P))"

  Ser2_1' "ALL G F P A .
           l(G,service_resolve(F,P,A)) BECAUSE
           l(A,delta(service_req(F,P))&guardedby(G))"
  Ser2_2' "ALL A S F P . EX G .
           l(A,service_available(S,F,P)) BECAUSE
           l(G,service_found(F,P,S,A))"
  Ser2_3' "ALL A F P . EX G .
           l(A,no_service_available(F,P)) BECAUSE
           l(G,no_service_found(F,P,A))"

  Ser2_2_3' "ALL A S F P . EX G .
           l(A,service_available(S,F,P)|
            no_service_available(F,P)) BECAUSE
           l(G,service_resolve(F,P,A))"

end

mob_adtl_services_G2.thy

mob_adtl_services_G2 = mob_adtl_services_G1 +

rules

  Ser2_5  "ALL G F P A . EX S .
           l(G,service_resolve(F,P,A)) LEADSTO
           l(G,service_found(F,P,S,A)|no_service_found(F,P,A))"
  Ser2_6  "ALL G F P S A .
           co l(G,service_found(F,P,S,A)-->
            satisfy(consCNL(S,nilCNL),
            consP(service_profile(F,P,A),nilP)))"
  Ser2_7  "ALL F F P A .
           co l(G,no_service_found(F,P,A)-->
            unsatisfiable(consP(service_profile(F,P,A),nilP)))"
  Ser2_5' "ALL G F P S A .
           l(G,service_found(F,P,S,A)|no_service_found(F,P,A))
           BECAUSE l(G,service_resolve(F,P,A))"

end
mob_adtl_services_refModel.thy

(* service discovery reference model *)

mob_adtl_services_refModel = mob_adtl_services_A1 +
    mob_adtl_services_G2 +
    mob_adtl_services_C2 +

end
mob_adtl_mobDevices_support.thy

(* everything is needed by mob_adtl theory for mobile devices *)

mob_adtl_mobDevices_support = mob_adtl_support +

consts

    new :: [component_name,profile] => o
    exit :: [component_name,profile] => o
    reachable :: component_name => o

end

mob_adtl_mobDevices_G.thy

(* specification of the first level of the PC meeting case study : guardians' behaviour *)

mob_adtl_mobDevices_G = mob_adtl_refModel +
    mob_adtl_mobDevices_support +

rules

Gmd1 "ALL G A O P T .
    l(G,mobReq(A,O,P,dest(T))) BECUASE
    l(G,moving(A)&satisfy(consCNL(T,nilCNL),P)&reachable(T))"

Gmd2 "ALL G A O P D .
    co_l(G,toBeVetoed(mob(A,O,P),D)--guarding(A))"

Gmd3 "ALL A O P D T1 T2 .
    co_l(G,delta(toBeVetoed(mob(A,O,P),D))&(D==dr)&
        reachable(T1)&satisfy(consCNL(T2,nilCNL),P)-->
        (T1==T2))"

SR1 "ALL G T P . l(G,delta(new(T,P))) LEADSTOC
    l(G,reachable(T)&satisfies(T,P))"

SR2 "ALL G T P . (l(G,delta(exit(T,P)))) LEADSTOC
    l(G,-reachable(T)&~satisfies(T,P))"

SR3 "ALL G T P . UNLESS(l(G,new(T,P)),l(G,exit(T,P)))"

end

mob_adtl_mobDevices_refModel.thy

(* mob_adtl reference model for mobile devices *)

mob_adtl_mobDevices_refModel = mob_adtl_refModel +

end
mob_adtl_migAgents_support.thy

(* everything is needed by the PC meeting case study *)
mob_adtl_migAgents_support = mob_adtl_support +

consts

meeting :: profile_item

locked :: component_name => o
wantToLeave :: o
leaving :: component_name => o

populationEmpty :: o
moved :: component_name => o

end

mob_adtl_migAgents_A.thy

(* specification of the second level of the PC meeting case study : agents' behaviour *)

mob_adtl_migAgents_A = mob_adtl_prof_name_refModel +
    mob_adtl_mobDevices_refModel +
    mob_adtl_migAgents_support +

rules

Amig "ALL A P H. EX T .
co_l(A,moveTo(P,H)-->P=consP(name(T),nilP))"

end

mob_adtl_migAgents_C.thy

(* specification of the second level of the PC meeting case study : coordination rules *)

mob_adtl_migAgents_C = mob_adtl_prof_name_refModel +
    mob_adtl_mobDevices_refModel +
    mob_adtl_migAgents_support +

rules

Cmigl "ALL G T A . EX H .
l(G,delta(locked(T))&guarding(A)) LEADSTO
l(A,delta(moveTo(consP(name(T),nilP),H))&guardedby(G))"

Cmigl' "ALL A H G . EX T .
l(A,moveTo(consP(name(T),nilP),H)&guardedby(G))
BECAUSE
l(G,delta(locked(T))))"

Cpo "ALL A T G H .
l(A,delta(guardedby(T))&history(consCNL(G,H)) LEADSTO
l(G,delta(moved(A))))"

end
mob_adtl_migAgents_G.thy

(* specification of the second level of the PC meeting case study: guardians' behaviour *)

mob_adtl_PCmeeting_G = mob_adtl_prof_name_refModel +
    mob_adtl_mobDevices_refModel +
    mob_adtl_migAgents_support +

rules

Asm1 "ALL G . EX T . co_l(G,wantToLeave->
    (new(T,consP(meeting,nilP)) &
    ~exit(T,consP(meeting,nilP))))"

Asm2 "ALL G G1 . co_l(G,wantToLeave->-leaving(G1))"

Gmig1 "ALL G A . EX T .
    l(G,delta(wantToLeave) & guarding(A)) LEADSTO
    l(T,leaving(G)) & l(G,delta(locked(T)) & guarding(A))"

Gmig2 "ALL T . EX G .
    l(T,delta(exit(G,P)) & leaving(G)) LEADSTO
    l(T,-leaving(G))"

Gmig3 "ALL G T . INIT l(T,-leaving(G))"

Gmig4 "ALL G T . INIT l(G,-locked(T))"

Gmig5 "ALL G A O P . co_l(G,mobReq(A,O,P,i)->-moving(A))"

Gmig6 "ALL G G1 .
    l(G,exit(G1,consP(meeting,nilP))) BECAUSE
    l(G1,populationEmpty)"

Gmig7 "ALL G T . co_l(G,delta(locked(T)))->reachable(T)"

Gmig8 "ALL G T . co_l(G,delta(locked(T)))->
    satisfy(consCNL(T,nilCNL),consP(meeting,nilP))"

Gmig9 "ALL G A O T D .
    co_l(G,-toBeVetoed(mob(A,O,consP(name(T),nilP)),D))"

Gmig1' "ALL G T . l(G,locked(T)) BECAUSE l(T,leaving(G))"

Gmig1'' "ALL G T . l(T,leaving(G)) BECAUSE l(G,locked(T))"

Gmig2' "ALL T G .
    UNLESS l(T,leaving(G)) , l(T,delta(exit(G,P)))"

Gpo1 "ALL G A .
    co_l(G,populationEmpty->-guarding(A)|moved(A))"

Gpo2 "ALL G A . INIT l(G,-moved(A))"

end

mob_adtl_migAgents_refModel.thy

(* reference model for the PC meeting case study *)

mob_adtl_migAgents_refModel = mob_adtl_migAgents_A +
    mob_adtl_migAgents_C + mob_adtl_migAgents_G +

end
Appendix B

This appendix contains the collection of all the proofs that have been performed using Mark. The proofs are presented in their completeness, as they come from the .ML files in which they are organized; an insight on the intermediate proof states and some comments are given to ease their comprehension. For a collection of all the theories based on which these proofs are based see Appendix A.

The logic theories

PC.ML

(* Iff implies Imp
 (A<->B)-->((A-->B)
 (A<->B)-->((B-->A)*)
 fun IffImp_tac(n) = (n
 (resolve_tac [impI] n) THEN
 ((eresolve_tac [IffE_left] n)
 ORELS
 (eresolve_tac [IffE_right] n))
);

(* Iff elimination tactic attacks implication goals trying to
 resolve their iff version*)
 fun IffE_tac(AXIOMS,n) = (n
 ((resolve_tac [IffE_right] n)
 THEN
 ((assume_tac n)
 ORELS
 (resolve_tac AXIOMS n)))
 ORELS
 ((resolve_tac [IffE_left] n)
 THEN
 ((assume_tac n)
 ORELS
 (resolve_tac AXIOMS n)))
);

Goal "F3<-(F1&F2)==>F3==>-F1";
by (resolve_tac [impI] 1);
by (resolve_tac [conjunct1] 1);
by (resolve_tac [mp] 1);
by (resolve_tac [IffE_right] 1);
by (assume_tac 1);
by (assume_tac 1);
qed "lemma1";

Goal "F3<-(F1&F2)==>F3==>-F2";
by (resolve_tac [impI] 1);
by (resolve_tac [conjunct2] 1);
by (resolve_tac [mp] 1);
by (resolve_tac [IffE_right] 1);
by (assume_tac 1);
by (assume_tac 1);
qed "lemma2";

Goal "F3<-(F1&F2) == -F3<-(~F1|~F2)";
by (fold_goals_tac [de_morgan1]);
by (resolve_tac [NotIffFN] 1);
by (assume_tac 1);
qed "lemma3";

Goal "F3<->F1-->F2==>F3-->~F2-->~F1";
by (resolve_tac [impI] 1);
by (resolve_tac [ContrapositionFN] 1);
by (resolve_tac [mp] 1);
by (resolve_tac [iffE_right] 1);
by (assume_tac 1);
by (assume_tac 1);
qed "lemma4";

Goal "F4<->F1-->F3 == F4<->(F1-->F2)&(F2-->F3)";
by (resolve_tac [iffI] 1);
by (fold_goals_tac [iffTrans]);
by (resolve_tac [iffE_right] 1);
by (assume_tac 1);
by (resolve_tac [iffE_left] 1);
by (assume_tac 1);
qed "lemma5";

Goal "((A-->B)&(C-->D))-->((A&C)-->B&D))";
by (Auto_tac);
qed "lemma6";

Goal "F3<->F1|F2==>~F1-->~F2-->~F3";
by (rewrite_goals_tac [Imp_def]);
by (Simp_tac 1);
by (resolve_tac [Or_rotate] 1);
by (Simp_tac 1);
by (resolve_tac [Or_rotate] 1);
by (Simp_tac 1);
by (fold_goals_tac [Imp_def]);
by (eresolve_tac [iffE_right] 1);
qed "lemma7";

Goal "(A&B)-->C==>A-->B-->C";
by (rewrite_goals_tac [imp_def]);
by (rewrite_goals_tac [de_morgan1]);
by (Asm_full_simp_tac 1);
qed "lemma9_1";

Goal "(A-->B-->C)||(A&B)-->C";
by (rewrite_goals_tac [imp_def]);
by (rewrite_goals_tac [de_morgan1]);
by (Asm_full_simp_tac 1);
qed "lemma9_2";

fun lemma9_2tac(n) = 
  (resolve_tac [lemma9_2] n) THEN
  (resolve_tac [impI] n) THEN
  (resolve_tac [impI] n)
);

Goal "((A&B)-->C)<->(A-->B-->C)";
by (rewrite_goals_tac [iff_def]);
by (resolve_tac [conjI] 1);
by (resolve_tac [impI] 1);
by (resolve_tac [lemma9_1] 1);
by (assume_tac 1);
by (resolve_tac [impI] 1);
by (resolve_tac [lemma9_2] 1);
by (assume_tac 1);
qed "lemma9";

Goal "((A&B)<->C)-->A-->C-->B";
by (Auto_tac);
qed "lemma10";

Goal "((A&C)<->B)-->A-->B";
by (resolve_tac [impI] 1);
by (resolve_tac [ContrapositionNP] 1);
by (Simp_tac 1);
by (resolve_tac [lemma1] 1);
by (resolve_tac [SwapIff] 1);
by (assume_tac 1);
qed "lemma11";

Goal "A|B<->B|A";
by (resolve_tac [iffI] 1);
by (resolve_tac [Or_rotate] 1);
by (assume_tac 1);
by (resolve_tac [Or_rotate] 1);
by (assume_tac 1);
qed "lemma12";

Goal "(A&B)<->A";
by (resolve_tac [impI] 1);
by (resolve_tac [mp] 1);
by (resolve_tac [lemma1] 1);
by (assume_tac 2);
by (resolve_tac [IffSimm] 1);
qed "lemma13";

Goal "(A&B)<->B";
by (resolve_tac [impI] 1);
by (resolve_tac [mp] 1);
by (resolve_tac [lemma2] 1);
by (assume_tac 2);
by (resolve_tac [IffSimm] 1);
qed "lemma14";

Goal "((A-->C)|((B-->C))-->((A&B)-->(A&C))"
by (Auto_tac);
qed "lemma15";

Goal "A<->((A&B)|(A~B))";
by (Auto_tac);
qed "lemma16";

Goal "(A|B)&(C|D)<->(A&C)|((B&C)|(A&D)|(B&D))";
by (Auto_tac);
qed "lemma17";

Goal "(A|B)&(C|D)<->(A&C)|((A&D)|(B&C)|(B&D))";
by (Auto_tac);
qed "lemma18";

Goal "(A|B)&(C|D)<->((C&A)|(D&A)|(C&B)|(D&B))";
by (Auto_tac);
qed "lemma19";

Goal "A<->((B&!A) | (B&A))";
by (Auto_tac);
qed "lemma20";

Goal "A<->(A|B)";
by (Auto_tac);
qed "lemma21";

Goal "A<->(B|A)";
by (Auto_tac);
qed "lemma22";

Goal "A<->(B|A|C)"
by (Auto_tac);
qed "lemma23";

Goal "A<->(B|A|C|D)"
by (Auto_tac);
qed "lemma24";

Goal "A<->(B|A|C|D|A|E)"
by (Auto_tac);
qed "lemma25";

Goal "A<->(B|A|C|D|E|A)"
by (Auto_tac);
qed "lemma26";

val PC_RULES =
[lemma1, lemma2, lemma3, lemma4, lemma5, lemma6, lemma7, lemma8, lemma9, lemma10, lemma11, lemma12, lemma13, lemma14, lemma15, lemma16, lemma17, lemma18, lemma19, lemma20, lemma21, lemma22];
DSL_support.ML

(* some derived properties for the equality relation *)

Goal ":-eq(A,C)-->eq(A,B)
by (fold_goals_tac [de_morgan1]);
by (resolve_tac [ContrapositionNP] 1);
by (Simp_tac 1);
by (resolve_tac [eqTran] 1);
ged "NoteqTran";

DSL.ML

(* from 1 to co_l *)
fun 12co_l_tac(n) = (  
  ((resolve_tac [NotNotE_mmf1_Iff_mf2] n) THEN  
    (resolve_tac [NotIffNP] n) THEN  
    (fold_goals_tac [co_l_def]))
ORELSE
  ((resolve_tac [NotNotE_mf1_Iff_mf2] n) THEN  
    (resolve_tac [NotIffNP] n) THEN  
    (fold_goals_tac [co_l_def]))
ORELSE
  ((resolve_tac [NotNotE_mf1_Imp_mf2] n) THEN  
    (resolve_tac [ContrapositionNP] n) THEN  
    (fold_goals_tac [co_l_def]))
ORELSE
  ((resolve_tac [NotNotE_mf1_Imp_mf2] n) THEN  
    (resolve_tac [ContrapositionNP] n) THEN  
    (fold_goals_tac [co_l_def]))
ORELSE
  ((resolve_tac [NotNotE_mf1_Imp_mf2ANDmf3] n) THEN  
    (resolve_tac [ContrapositionNP] 1) THEN  
    (rewrite_goals_tac [de_morgan1]) THEN  
    (fold_goals_tac [co_l_def]))
ORELSE
  ((resolve_tac [NotNotE_mf1_Imp_mf2ANDmf3] n) THEN  
    (resolve_tac [ContrapositionNP] n) THEN  
    (rewrite_goals_tac [de_morgan2]) THEN  
    (fold_goals_tac [co_l_def]))
ORELSE
  ((resolve_tac [NotNotE_mf1_Imp_mf2ANDmf3] n) THEN  
    (resolve_tac [ContrapositionNP] n) THEN  
    (fold_goals_tac [co_l_def]))
ORELSE
  ((resolve_tac [NotNotE_mf1_Iff_f2] n) THEN  
    (resolve_tac [NotIffNP] n) THEN  
    (fold_goals_tac [co_l_def]))
ORELSE
  ((resolve_tac [ContrapositionNP] 1) THEN  
    (Simp_tac 1) THEN  
    (rewrite_goals_tac [co_l_def]) THEN  
    (Simp_tac 1))
)

(* resolution with K via one application of modus ponens  
from a co_l(M,F1)-->co_l(M,F2)  
to a co_l(M,F2-->F1) *)
fun K_tac_simple(n) = 
  (resolve_tac [mp] n) THEN
  (resolve_tac [K] n)
);

(* resolution with K via two applications of modus ponens
from a co_l(M,F1)
to a co_l(M,F2-->F1)
with F2 instantiated using resolution on AXIOM *)

fun K_tac_no_asm(AXIOM,n) = 
  (resolve_tac [mp] n) THEN
  (resolve_tac [AXIOM] [n+1]) THEN
  (resolve_tac [mp] n) THEN
  (resolve_tac [K] n)
);

fun K_tac_asm(AXIOM,n) = 
  (resolve_tac [mp] n) THEN
  (resolve_tac [AXIOM] [n+1]) THEN
  (((assume_tac (n+1)) THEN
    (resolve_tac [mp] n) THEN
    (resolve_tac [K] n))
  ORELSE
  (((resolve_tac [mp] n) THEN
    (resolve_tac [K] n))))
);

(* some DSL properties derived from DSL axioms *)

val DSL_AXIOMS = [K,DSL1,DSL2,Nec];

Goal "F1-->F2==>l(M,F1)--->l(M,F2)";
  by (12co_l_tac(1));
  by (K_tac_simple(1));
  by (resolve_tac [Nec] 1);
  by (eresolve_tac [ContrapositionPN] 1);
qed "l_ref";

Goal "F2-->F1==>co_l(M,F2)--->co_l(M,F1)";
  by (K_tac_simple(1));
  by (eresolve_tac [Nec] 1);
qed "co_l_ref";

(* resolution with l_ref via modus ponens *)

fun l_ref_tac(n) = 
  (((resolve_tac [l_ref] n))
  ORELSE
  ((resolve_tac [mp] n) THEN
    (resolve_tac [l_ref] n)))
);

(* resolution with co_l_ref via modus ponens *)

fun co_l_ref_tac(n) = 
  (((resolve_tac [co_l_ref] n))
  ORELSE
  ((resolve_tac [mp] n) THEN
    (resolve_tac [co_l_ref] n)))
);

val DSL_AXIOMS = DSL_AXIOMS@[l_ref,co_l_ref];
Goal "co_1(M,F)<->co_1(M,co_1(M,F))";
by (resolve_tac [IfF] 1);
(* --> *)
by (K_tac_simple(1));
by (co_1_ref_tac(1));
by (resolve_tac DSL_AXIOMS 2);
by (resolve_tac [impI] 1);
by (IfF_tac(DSL_AXIOMS,1));
(* --> *)
by (K_tac_simple(1));
by (co_1_ref_tac(1));
by (resolve_tac DSL_AXIOMS 2);
by (resolve_tac [impI] 1);
by (IfF_tac(DSL_AXIOMS,1));
qed "assiom4";

val DSL_AXIOMS = DSL_AXIOMS@[assiom4];

Goal "l(M,F)<->l(M,1(M,F))";
by (l2co_1_tac(1));
by (resolve_tac DSL_AXIOMS 1);
qed "D1";

val DSL_AXIOMS = DSL_AXIOMS@[D1];

Goal "F3<-(F1&F2)==>l(M,F3)==>l(M,F1) & l(M,F2)";
by (resolve_tac [impI] 1);
by (resolve_tac [conjI] 1);
(* F3<-(F1&F2)==>l(M,F3) ==> l(M,F1) *)
by (resolve_tac [mp] 1);
by (assume_tac 2);
back();
by (l_ref_tac(1));
by (eresolve_tac PC_RULES 1);
(* F3<-(F1&F2)==>l(M,F3) ==> l(M,F1) *)
by (resolve_tac [mp] 1);
by (assume_tac 2);
back();
by (l_ref_tac(1));
by (eresolve_tac PC_RULES 1);
qed "D2";

val DSL_AXIOMS = DSL_AXIOMS@[D2];

Goal "F3<-(F1&F2)==>co_1(M,F3)==>(co_1(M,F1) & co_1(M,F2))";
by (resolve_tac [impI] 1);
by (resolve_tac [conjI] 1);
by (co_1_ref_tac(1));
by (assume_tac 2);
by (eresolve_tac PC_RULES 1);
by (co_1_ref_tac(1));
by (assume_tac 2);
by (eresolve_tac PC_RULES 1);
qed "co_D2";

val DSL_AXIOMS = DSL_AXIOMS@[co_D2];

Goal "F3<-(F1|F2)==>l(M,F3)<->(l(M,F1) | l(M,F2))";
by (resolve_tac [IfF] 1);
(* --> *)
by (12co_l_tac(1));
by (rewrite_goals_tac [Imp_def]);
by (Simp_tac 1);
by (fold_goals_tac [Imp_def]);
by (resolve_tac [impI] 1);
by (K_tac_simple(1));
by (resolve_tac [mp] 1);
by (assume_tac 2);
back();
by (K_tac_simple(1));
by (resolve_tac [Nec] 1);
by (eresolve_tac [lemma7] 1);  (* <-- *)
by (rewrite_goals_tac [Imp_def]);
by (rewrite_goals_tac [de_morgan2]);
by (rewrite_goals_tac [OrAnd_distr]);
by (fold_goals_tac [Imp_def]);
by (resolve_tac [conjI] 1);
by (l_ref_tac(1));
by (Asm_simp_tac 1);
by (l_ref_tac(1));
by (Asm_simp_tac 1);
qed "Or_l";

Goal "l(M,F1,F2) == (l(M,F1) | l(M,F2))";
by (resolve_tac [logicLemma1] 1);
by (resolve_tac [Or_l] 1);
by (Simp_tac 1);
qed "Or_l_eq";

val DSL_AXIOMS = DSL_AXIOMS@[Or_l];

Goal "F3<-(F1&F2)==>(co_l(M,F1)&co_l(M,F2))-->co_l(M,F3)";
by (SELECT_GOAL (rewrite_goals_tac [co_l_def]) 1);
by (rewrite_goals_tac [Imp_def]);
by (Simp_tac 1);
by (resolve_tac [Or_rotate] 1);
by (Simp_tac 1);
by (resolve_tac [Or_rotate] 1);
by (Simp_tac 1);
by (resolve_tac [Imp_def2] 1);
by (iffE_tac(DSL_AXIOMS,1));
by (fold_goals_tac [de_morgan1]);
by (eresolve_tac [NotiffPN] 1);
qed "And_co_l";

val DSL_AXIOMS = DSL_AXIOMS@[And_co_l];

Goal "F3<-(F1-->F2)==>co_l(M,F3)-->(l(M,F1)--->l(M,F2))";
by (resolve_tac [impI] 1);
by (12co_l_tac(1));
by (K_tac_simple(1));
by (co_l_ref_tac(1));
by (assume_tac 2);
by (eresolve_tac PC_RULES 1);
qed "D3";

val DSL_AXIOMS = DSL_AXIOMS@[D3];

Goal "co_l(M,F)-->l(M,True)--->l(M,F)";
by (resolve_tac [D3] 1);
by (Simp_tac 1);
qed "D4";

val DSL_AXIOMS = DSL_AXIOMS@[D1];

Goal "\( \text{co}_1(\text{M}, (\text{L}(\text{M}, \text{F}) \leftrightarrow \text{F}))) \);
by (co_1_ref_tac(1));
by (resolve_tac DSL_AXIOMS 2);
by (resolve_tac [impI] 1);
by (l2co_1_tac(1));
by (resolve_tac [SwapIfI] 1);
by (Simp_tac 1);
qed "D5";

val DSL_AXIOMS = DSL_AXIOMS@[D5];

Goal "\( \text{F}4 \leftrightarrow (\text{F}1 \rightarrow \text{F}3) \rightarrow (\text{co}_1(\text{M}, \text{F}1 \rightarrow \text{F}2) \land \text{co}_1(\text{M}, \text{F}2 \rightarrow \text{F}3)) \rightarrow \text{co}_1(\text{M}, \text{F}4) \)"
by (resolve_tac [And_co_1] 1);
by (eresolve_tac PC_RULES 1);
qed "D6";

val DSL_AXIOMS = DSL_AXIOMS@[D6];

Goal "\( \text{co}_1(\text{M}, (\text{L}(\text{M}, \text{F}1) \rightarrow \text{F}1) \land (\text{L}(\text{M}, \text{F}2) \rightarrow \text{F}2)) \rightarrow \text{co}_1(\text{M}, (\text{L}(\text{M}, \text{F}1) \land (\text{L}(\text{M}, \text{F}2)) \rightarrow (\text{F}1 \& \text{F}2)) \)"
by (K_tac_simple(1));
by (resolve_tac [Nec] 1);
by (resolve_tac [lemma6] 1);
qed "DSLlemma1";

Goal "\( \text{co}_1(\text{M}, (\text{L}(\text{M}, \text{F}1) \land (\text{L}(\text{M}, \text{F}2)) \rightarrow (\text{L}(\text{M}, \text{F}1 \& \text{F}2)) \)"
by (resolve_tac [D6 RS mp] 1);
by (Simp_tac 1);
by (resolve_tac [conjI] 1);
by (resolve_tac [mp] 1);
by (resolve_tac [DSLlemma1] 1);
by (resolve_tac [mp] 1);
by (resolve_tac [And_co_1] 1);
by (resolve_tac [IfI] 1);
by (resolve_tac [impI] 1);
by (assume_tac 1);
by (Simp_tac 1);
by (resolve_tac [conjI] 1);
by (co_1_ref_tac(1));
by (resolve_tac [D5] 2);
by (resolve_tac [impI] 1);
by (IfI_tac(DSL_AXIOMS,1));
by (co_1_ref_tac(1));
by (resolve_tac [D5] 2);
by (resolve_tac [impI] 1);
by (IfI_tac(DSL_AXIOMS,1));
by (co_1_ref_tac(1));
by (resolve_tac [D5] 2);
by (resolve_tac [impI] 1);
by (IfI_tac(DSL_AXIOMS,1));
qed "D8";

val DSL_AXIOMS = DSL_AXIOMS@[D8];

Goal "\( |\text{co}_1(\text{M}, \text{F}1 \rightarrow \text{F}2); \text{co}_1(\text{M}, \text{F}2 \rightarrow \text{F}1)| \rightarrow \text{co}_1(\text{M}, \text{F}1 \leftrightarrow \text{F}2) |) \)";
by (resolve_tac [mp] 1);
by (resolve_tac [And_co_1] 1);
by (resolve_tac [IffI] 1);
by (resolve_tac [impI] 2);
by (SELECT_GOAL (rewrite_goals_tac [Iff_def]) 2);
by (assume_tac 2);
by (Simp_tac 1);
by (resolve_tac [conjI] 1);
by (assume_tac 1);
by (assume_tac 1);
qed "IffI_co_1";

val DSL_AXIOMS = DSL_AXIOMS@[IffI_co_1];

Goal "(co_1(M,A<->B)&co_1(M,A))-->co_1(M,B)";
by (resolve_tac [mp] 1);
by (resolve_tac [IffE_right] 1);
by (resolve_tac [SwapIff] 1);
by (resolve_tac PC_RULES 1);
back();
by (resolve_tac [impI] 1);
by (K_tac_simple(1));
by (co_1_ref_tac(1));
by (assume_tac 2);
by (Simp_tac 1);
qed "Iff_co_11";

val DSL_AXIOMS = DSL_AXIOMS@[Iff_co_11];

Goal "(co_1(M,A<->B)&co_1(M,B))-->co_1(M,A)";
by (resolve_tac [mp] 1);
by (resolve_tac [IffE_right] 1);
by (resolve_tac [SwapIff] 1);
by (resolve_tac PC_RULES 1);
back();
by (resolve_tac [impI] 1);
by (K_tac_simple(1));
by (co_1_ref_tac(1));
by (assume_tac 2);
by (Simp_tac 1);
qed "Iff_co_12";

val DSL_AXIOMS = DSL_AXIOMS@[Iff_co_12];

Goal "(co_1(M,A<->B)&1(M,A))-->1(M,B)";
by (resolve_tac [mp] 1);
by (resolve_tac [IffE_right] 1);
by (resolve_tac [SwapIff] 1);
by (resolve_tac PC_RULES 1);
back();
by (resolve_tac [impI] 1);
by (resolve_tac [mp] 1);
by (resolve_tac [D3] 1);
by (resolve_tac [IffSimm] 1);
by (co_1_ref_tac(1));
by (assume_tac 2);
by (Simp_tac 1);
qed "Iff_co_1l_weak";

val DSL_AXIOMS = DSL_AXIOMS@[Iff_co_1l_weak];
Goal "(co_1(M,A<->B) & l(M,B)) --> l(M,A)";
by (resolve_tac [mp] 1);
by (resolve_tac [IffE_right] 1);
by (resolve_tac [SwapIff] 1);
by (resolve_tac PC_RULES 1);
back();
by (resolve_tac [impI] 1);
by (resolve_tac [mp] 1);
by (resolve_tac [D3] 1);
by (resolve_tac [IffSimm] 1);
by (co_l_ref_tac(1));
by (assume_tac 2);
by (Simp_tac 1);
qed "iff_co_12_weak";

val DSL_AXIOMS = DSL_AXIOMS@[iff_co_12_weak];

Goal "l(M,F) = l(M,F & G) | l(M,F & ~G)"
by (resolve_tac [logicLemma1] 1);
by (resolve_tac [Or_l] 1);
by (resolve_tac [lemma16] 1);
qed "DSLlemma2_1";

Goal "l(M,F) = l(M,G & F) | l(M,~G & F)"
by (resolve_tac [logicLemma1] 1);
by (resolve_tac [Or_l] 1);
by (resolve_tac [lemma20] 1);
qed "DSLlemma2_2";

val DSL_AXIOMS = DSL_AXIOMS@[DSLlemma2_1, DSLlemma2_2];

Goal "l(M,F & G) | l(M,F & ~G) --> l(M,F)"
by (resolve_tac [mp] 1);
by (assume_tac 2);
by (resolve_tac [IffE_right] 1);
by (resolve_tac [SwapIff] 1);
by (resolve_tac [Or_l] 1);
by (resolve_tac [lemma16] 1);
qed "DSLlemma3";

val DSL_AXIOMS = DSL_AXIOMS@[DSLlemma3];

Goal "(EX X . l(A,P(X))) --> (l(A,EX X . P(X)))"
by (resolve_tac [impI] 1);
by (eresolve_tac [exE] 1);
by (resolve_tac [mp] 1);
by (resolve_tac [l_ref] 1);
by (assume_tac 2);
by (resolve_tac [impI] 1);
by (resolve_tac [exI] 1);
by (assume_tac 1);
qed "ex1_lex";

/*Goal "l(A,EX X . P(X)) --> (EX X . l(A,P(X)))";
qed "leX_ex1";*/

val DSL_AXIOMS = DSL_AXIOMS@[ex1_lex];

Goal "l(A,P(X)) --> l(A,EX X . P(X))"
by (resolve_tac [l_ref] 1);
by (resolve_tac [impI] 1);
by (resolve_tac [exI] 1);
by (assume_tac 1);
qed "l_LEX";

val DSL_AXIOMS = DSL_AXIOMS@[l_LEX];
DSTL.ML

val DSTL_AXIOMS = DSL_AXIOMS@[LcI,BcI,LI,BI,UI,InI,SI,SE,LTR,BTR,UC,LSW,LPD,LCC,BSW,BPD,BCC,LcSW,LcPD,LcCC,BcSW,BcPD,BcCC,Notif,Conf,I1,I2];

(* some tactics *)

(* tactic to apply sequentially a list of resolutions
   does nothing if L is nil *)
fun rec_res_tac(L,n) =
  case L of
    [] => all_tac |
    A::L1 => (resolve_tac [A] n) THEN rec_res_tac(L1,n)
);

(* transitivity: apply transitivity theorem TR_RULE
   AXIOMS_FirstPrem if not empty is supposed to be enough to
   solve the first premise *)
fun Tran_tac(TR_RULE,AXIOMS_FirstPrem, AXIOMS_SecondPrem,n) = ( 
  (resolve_tac [TR_RULE] n) THEN
  (case AXIOMS_FirstPrem of
   A::L => (rec_res_tac(AXIOMS_FirstPrem,n)) THEN
     (rec_res_tac(AXIOMS_SecondPrem,n)) |
   [] => (rec_res_tac(AXIOMS_SecondPrem,n+1)))
);

(* tactic for *SW theorems
   AXIOMS_FirstPrem and AXIOMS_SecondPrem if not empty are
   supposed to be enough to solve the first or the second
   premise respectively*)
fun SW_tac(SW_RULE,AXIOMS_FirstPrem, AXIOMS_SecondPrem,
          AXIOMS_ThirdPrem,n) = ( 
  (resolve_tac [SW_RULE] n) THEN
  (case AXIOMS_FirstPrem of
   A1::L1 => rec_res_tac(AXIOMS_FirstPrem,n) THEN
     ((case AXIOMS_SecondPrem of
      A2::L2 => rec_res_tac(AXIOMS_SecondPrem,n) THEN
        rec_res_tac(AXIOMS_ThirdPrem,n) |
        [] => rec_res_tac(AXIOMS_ThirdPrem,n+1))) |
      [] => (case AXIOMS_SecondPrem of
        A3::L3 => rec_res_tac(AXIOMS_SecondPrem,n+1) THEN
          rec_res_tac(AXIOMS_ThirdPrem,n+1) |
          [] => rec_res_tac(AXIOMS_ThirdPrem,n+2)))
  ));

fun SW_tac_asm(SW_RULE,AXIOMS_FirstPrem, AXIOMS_ThirdPrem,n) = ( 
  (resolve_tac [SW_RULE] n) THEN
  (case AXIOMS_FirstPrem of
   A1::L1 => rec_res_tac(AXIOMS_FirstPrem,n) THEN
    ((assume_tac n) THEN
     rec_res_tac(AXIOMS_ThirdPrem,n)) |
    [] => ((assume_tac (n+1)) THEN
     rec_res_tac(AXIOMS_ThirdPrem,n+1)))
  );

(* leads to by cases: apply a by cases reasoning on the premises
   of a leads_to goal *)
fun LTbyCases_tac(n) = ( 
  (resolve_tac [LSW] n) THEN

(resolve_tac [IffE_right] n) THEN
(resolve_tac [lemma16] n) THEN
(resolve_tac [Imp_refl] (n+1)) THEN
(resolve_tac [LPD] n)
);

val ger_Cor1 = (LSW RS (LSW RSN (2,LPD))) RSN (2,LTR);
val DSTL_AXIOMS = DSTL_AXIOMS@[ger_Cor1];

Goal "[[ F1 LEADSTO (G1|G2); G1 LEADSTO F2]]==>
F1 LEADSTO (F2|G2)"
by (resolve_tac [ger_Cor1] 1);
by (assume_tac 1);
by (resolve_tac [Imp_refl] 4);
by (resolve_tac [LI] 4);
by (resolve_tac [LCI] 4);
by (Simp_tac 4);
by (assume_tac 2);
back();
by (Simp_tac 1);
by (Simp_tac 1);
qed "Cor1_1";

Goal "[[ F1 LEADSTO (G1|G2); G1 LEADSTO (F2_1|F2_2)];]]==>
F1 LEADSTO ((F2_1|F2_2)|G2)"
by (resolve_tac [Cor1_1] 1);
by (assume_tac 1);
by (assume_tac 1);
qed "Cor1_1lemma1";

Goal "[[F1 LEADSTO (G2|G1); G1 LEADSTO F2)];]]==>
F1 LEADSTO (F2|G2)"
by (resolve_tac [ger_Cor1] 1);
by (assume_tac 1);
by (resolve_tac [Imp_refl] 1);
by (resolve_tac [LI] 1);
by (resolve_tac [LCI] 1);
by (Simp_tac 1);
by (assume_tac 2);
back();
by (Simp_tac 1);
by (Simp_tac 1);
qed "Cor1_2";

val DSTL_AXIOMS = DSTL_AXIOMS@[Cor1_1,Cor1_2];
val ger_Cor2 = (LSW RS LTR) RS (LSW RSN (2,LCC));
val DSTL_AXIOMS = DSTL_AXIOMS@[ger_Cor2];

Goal "[[F1 LEADSTO (G1&G2); G1 LEADSTO F2)];]]==>
F1 LEADSTO (F2&G2)"
by (resolve_tac [ger_Cor2] 1);
by (assume_tac 2);
by (Simp_tac 1);
by (assume_tac 2);
by (Simp_tac 1);
by (Simp_tac 2);
by (Simp_tac 1);
qed "Cor2_1";

Goal "[[ F1 LEADSTO (G2&G1); G1 LEADSTO F2]|] ==>
   F1 LEADSTO (F2&G2)");
by (resolve_tac [ger_Cor2] 1);
by (assume_tac 2);
by (Simp_tac 1);
by (assume_tac 2);
by (Simp_tac 1);
by (assume_tac 2);
by (Simp_tac 1);
by (Simp_tac 1);
by (Simp_tac 1);
qed "Cor2_2";

val DSTL_AXIOMS = DSTL_AXIOMS@[Cor2_1,Cor2_2];

val ger_BCor1 = (BSW RS (BSW RSN (2,BFD)) RSN (2,BTR));

val DSTL_AXIOMS = DSTL_AXIOMS@[ger_BCor1];

Goal "[[ F1 BECAUSE (G1|G2);G1 BECAUSE F2]|] ==>
   F1 BECAUSE (F2|G2)");
by (resolve_tac [ger_BCor1] 1);
by (assume_tac 1);
by (resolve_tac [Imp_refl] 4);
by (resolve_tac [BI] 4);
by (resolve_tac [BcI] 4);
by (Simp_tac 4);
by (assume_tac 2);
back();
by (Simp_tac 1);
by (Simp_tac 1);
qed "BCor1_1";

Goal "[[ F1 BECAUSE (G1|G2);G1 BECAUSE (F2_1|F2_2)|] ==>
   F1 BECAUSE ((F2_1|F2_2)|G2)");
by (resolve_tac [BCor1_1] 1);
by (assume_tac 1);
by (assume_tac 1);
qed "BCor1_1lem1";

Goal "[[ F1 BECAUSE (G2|G1);G1 BECAUSE F2)|] ==>
   F1 BECAUSE (F2|G2)");
by (resolve_tac [ger_BCor1] 1);
by (assume_tac 1);
by (resolve_tac [Imp_refl] 1);
by (resolve_tac [BI] 1);
by (resolve_tac [BcI] 1);
by (Simp_tac 1);
by (assume_tac 2);
back();
by (Simp_tac 1);
by (Simp_tac 1);
qed "BCor1_2";

val DSTL_AXIOMS = DSTL_AXIOMS@[BCor1_1,BCor1_2];

val ger_BCor2 = (BSW RS BTR) RS (BSW RSN (2,BCC));

val DSTL_AXIOMS = DSTL_AXIOMS@[ger_BCor2];
Goal "[| F1 BECAUSE (G1&G2);G1 BECAUSE F2|] ==> 
F1 BECAUSE (F2&G2)"
by (resolve_tac [ger_BCor2] 1);
by (assume_tac 2);
by (Simp_tac 1);
by (assume_tac 2);
by (Simp_tac 1);
by (assume_tac 2);
by (Simp_tac 1);
by (Simp_tac 1);
by (Simp_tac 1);
qed "BCor2_1";

Goal "[| F1 BECAUSE (G2&G1); G1 BECAUSE F2|] ==> 
F1 BECAUSE (F2&G2)"
by (resolve_tac [ger_BCor2] 1);
by (assume_tac 2);
by (Simp_tac 1);
by (assume_tac 2);
by (Simp_tac 1);
by (assume_tac 2);
by (Simp_tac 1);
by (Simp_tac 1);
by (Simp_tac 1);
qed "BCor2_2";

val DSTL_AXIOMS = DSTL_AXIOMS@[BCor2_1,BCor2_2];

Goal " l (M,F1) LEADSTO l (N,F2) ==> 
 l (M,F1) LEADSTO (l(M,F1)&l(N,F2))"
by (resolve_tac [LCC] 1);
by (resolve_tac [LI] 1);
by (resolve_tac [LcI] 1);
by (Simp_tac 1);
qed "DSTLlemma1";

Goal "(l(M,F1)&l(N,F2)) LEADSTO l(M,F1)"
by (resolve_tac [LSW] 1);
by (resolve_tac [lemma13] 1);
by (resolve_tac [LI] 1);
by (resolve_tac [LcI] 1);
by (Simp_tac 1);
qed "DSTLlemma2";

Goal "(l(M,F1)&l(N,F2)) LEADSTO l(N,F2)"
by (resolve_tac [LSW] 1);
by (resolve_tac [lemma14] 1);
by (resolve_tac [LI] 1);
by (resolve_tac [LcI] 1);
by (Simp_tac 1);
qed "DSTLlemma3";

Goal "F1 LEADSTO (F1|F2)"
by (resolve_tac [LSW] 1);
by (resolve_tac [Imp_refl] 1);
by (resolve_tac [LI] 1);
by (resolve_tac [LcI] 1);
by (Simp_tac 1);
qed "DSTLlemma4";

Goal "F1 LEADSTO (F2|F1)"
by (resolve_tac [LSW] 1);
by (resolve_tac [Imp_refl] 1);
by (resolve_tac [LI] 1);
by (resolve_tac [LcI] 1);
by (Simp_tac 1);
qed "DSTlemma5";

Goal "STABLE(l(A,P(X))) --> STABLE(l(A,EX X . P(X)))";
by (resolve_tac [impI] 1);
by (resolve_tac [SN] 1);
by (assume_tac 1);
by (resolve_tac [l_ref] 1);
by (resolve_tac [impI] 1);
by (resolve_tac [exI] 1);
by (assume_tac 1);
qed "stableEX_I";

Goal "(EX X . INIT(l(A,P(X)))) --> INIT(l(A,EX X . P(X)))";
by (resolve_tac [impI] 1);
by (resolve_tac [exE] 1);
by (resolve_tac [IN] 1);
by (assume_tac 1);
by (resolve_tac [l_ref] 1);
by (resolve_tac [impI] 1);
by (resolve_tac [exI] 1);
by (assume_tac 1);
qed "initEX_lemma";
**The model theories**

**mob_adtl_support.ML**

(* tactics to solve goals regarding component name lists for both the following tactics, if the given goal is actually a theorem it will be solved, otherwise in the proof state a subgoal will remain stating that a component name should be element of an empty list *)

(* to check if a component name belongs to a list *)

```ml
fun isInCNL_tac (n) = (;
    (resolve_tac [isInCNL_rule] n) THEN
    ((resolve_tac [disjI1] n) THEN
      (Simp_tac n)) ORELSE
    (resolve_tac [disjI2] n))
);
```

(* to check if a list is a sublist of another list *)

```ml
fun subCNL_tac(n) = (;
    (resolve_tac [subCNL_nil] n) ORELSE
    ((resolve_tac [subCNL_rule] n) THEN
      isInCNL_tac(n))
);
```
mob_adtl_refModel.ML

(* properties of the mob_adtl reference model *)

Goal "ALL A P H O . EX T D1 D2 D3 .
   l(A,delta(moveTo(P,H))&guardedby(O)) LEADSTO
   l(A,delta((guardedby(T)))|
   delta(exc(mob(A,O,P),D1))|
   delta(exc(mob(A,O,P),D2))|
   delta(exc(mob(A,O,P),D3))|
   delta(exc(mob(A,O,P),dr)))";
by (REPEAT(resolve_tac [allI] 1));
by (cut_facts_tac [Gm1] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Gm2] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Gm3] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Gm5] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Cm2] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Cm3] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Cm3] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Cm3] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Cm3] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Cm3] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [A3] 1);
(* from the will to move of A to the mobility request in O *)
(* exploiting relations between guardedby and guarding we can
deduce that (guarding(A)|moving(A)) will hold in the state in
which the request will arrive*)
by (resolve_tac [LTR] 1);
by (resolve_tac [LTBI] 1);
by (resolve_tac [U2S] 4);
by (cut_facts_tac [A3] 4);
by (REPEAT (dresolve_tac [spec] 4));
by (REPEAT (eresolve_tac [exE] 4));
by (assume_tac 4);
by (cut_facts_tac [A4] 4);
by (REPEAT (dresolve_tac [spec] 4));
by (REPEAT (eresolve_tac [exE] 4));
by (assume_tac 4);
by (cut_facts_tac [Cm1] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (assume_tac 1);
192
by (SW_tac (BSW, [l_ref, lemma14], [], [Imp_ref1], 1));
by (cut_facts_tac [Cm2'] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (assume_tac 1);
by (resolve_tac [l_ref] 1);
by (SELECT_GOAL (Auto_tac) 1);
(* from the mobility request in 0 to the outcome of the request *)
(* splitting the cases on the condition (guarding(A)|moving(A)) *)
by (SELECT_GOAL (rewrite_goals_tac [AndOr_distr]) 1);
by (SW_tac (LSW, [IffE_right, Or_I, IffSimm], [], [Imp_ref1], 1));
by (resolve_tac [LFD] 1);
(* the mobility request reaches the guardian when guarding(A) holds *)
(* the guardian starts serving the request, either it forwards it to another guardian or vetoes it *)
by (Tran_tac (LTR, [], [], 1));
by (assume_tac 1); (**)
by (resolve_tac [LFD] 1);
(* the guardian forwards the request *)
(* the destination guardian receives the request or another guardian vetoes it *)
by (Tran_tac (LTR, [], [], 1));
by (assume_tac 1); (**)
by (resolve_tac [LFD] 1);
(* the destination guardian receives the request and either accepts or vetoes it *)
by (Tran_tac (LTR, [], [], 1));
by (assume_tac 1); (**)
by (resolve_tac [LFD] 1);
(* the destination guardian accepts the request *)
by (SW_tac_asm (LSW, [Imp_ref1], [l_ref], 1));
by (SELECT_GOAL (Auto_tac) 1);
(* the destination guardian vetoes the request *)
by (SW_tac_asm (LSW, [Imp_ref1], [l_ref], 1));
by (SELECT_GOAL (Auto_tac) 1);
(* another guardian vetoes the request *)
by (SW_tac_asm (LSW, [Imp_ref1], [l_ref], 1));
by (SELECT_GOAL (Auto_tac) 1);
(* the mobility request reaches the guardian AND moving(A) holds *)
by (Tran_tac (LTR, [], [], 1));
by (assume_tac 1);
by (SW_tac_asm (LSW, [Imp_ref1], [l_ref], 1));
by (SELECT_GOAL (Auto_tac) 1);
qed "lemmaM1";
Goal "ALL A P H O . EX T D1 D2 D3 .
  1(A, delta(moveTo(P,H))&guardedby(O)) LEADSTO
  1(A, delta(guardedby(T)))
  1(A, delta(exc(mob(A,O,P),D1)))
  1(A, delta(exc(mob(A,O,P),D2)))
  1(A, delta(exc(mob(A,O,P),D3)))
  1(A, delta(exc(mob(A,O,P),dr))))";
by (REPEAT(resolve_tac [allI] 1));
by (cut_facts_tac [lemmaM1] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (REPEAT (resolve_tac [exI] 1));
by (SW_tac_asm(LSW,[Imp_refl],[1],1));
by (resolve_tac [IffE_right] 1);
by (SELECT_GOAL (rewrite_goals_tac [Or_l_eq]) 1);
by (Simp_tac 1);
qed "lemmaM2";

Goal "ALL A P H O . EX T D .
  leads_to(1(A, delta(moveTo(P,H))&guardedby(O)) LEADSTO
  1(A, delta(guardedby(T)))|delta(exc(mob(A,O,P),D))))";
by (REPEAT(resolve_tac [allI] 1));
by (cut_facts_tac [lemmaM2] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (REPEAT(TRY(eresolve_tac [LdisjE] 1) THEN
  (REPEAT (resolve_tac [exI] 1)) THEN
  (SW_tac_asm(LSW,[Imp_refl],[1_ref],1)) THEN
  (SELECT_GOAL (Auto_tac 1))));
qed "M";

Goal "ALL A T . EX P H G1 O St G2 D.
  1(A, guardedby(T)) BECAUSE
  1(A, delta(moveTo(P,H)))&
  1(G1, mobReq(A,O,P,St)&satisfy(consCNL(T,nilCNL),P))|
  1(G2, toBeVetoed(mob(A,T,P),D))";
by (REPEAT (resolve_tac [allI] 1));
by (cut_facts_tac [Cm2'] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Gm3_6'] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Gm2'] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Gm1'] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Cm1'] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (REPEAT (resolve_tac [exI] 1));
by (resolve_tac [BOR] 1);
by (resolve_tac [BCC] 2);
by (resolve_tac [BI] 3);
by (resolve_tac [BcI] 3);
by (resolve_tac [BOR] 1);
by (assume_tac 2);
by (resolve_tac [BTR] 1);
by (assume_tac 1);
by (assume_tac 1);
by (resolve_tac [BTR] 1);
by (SW_tac_asm (BSW, [1_ref, lemma13], [Imp_refl], 1));
by (SW_tac_asm (BSW, [1_ref, lemma13], [1_ref, lemma13], 1));
qed "M1";

Goal "ALL A T . EX O P St1 St2 G D1 .
1(A, guardedby(T)) BECAUSE
1(T, mobReq(A, O, P, St1)) &
1(O, mobReq(A, O, P, St2)) &
1(G, toBeVetoed(mob(A, T, P), D1))";
by (REPEAT (resolve_tac [allI] 1));
by (cut_facts_tac [Cm2'] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Gm3_6'] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Gm2'] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Gm1'] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (REPEAT (resolve_tac [exI] 1));
by (resolve_tac [BTR] 1);
by (assume_tac 1);
by (resolve_tac [BOr] 1);
by (assume_tac 1);
by (resolve_tac [BAnd] 1);
by (resolve_tac [BI] 1);
by (resolve_tac [BcI] 1);
by (resolve_tac [BTR] 1);
by (assume_tac 1);
by (SW_tac_asm (BSW, [1_ref, lemma13], [1_ref, lemma13], 1));
qed "A1";

Goal "ALL A O P D . EX G St .
1(A, exc(mob(A, O, P), D)) BECAUSE
1(G, mobReq(A, O, P, St)) & 1(G, toBeVetoed(mob(A, O, P), D))";
by (REPEAT (resolve_tac [allI] 1));
by (cut_facts_tac [Cm3'] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Gm1_2_3'] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (REPEAT (resolve_tac [exI] 1));
by (resolve_tac [BTR] 1);
by (assume_tac 1);
by (resolve_tac [BAnd] 1);
by (assume_tac 1);
by (resolve_tac [BI] 1);
by (resolve_tac [BcI] 1);
qed "lemmaM2_1";

Goal "ALL A O P D . EX H .
1(O, mobReq(A, O, P, i)) BECAUSE 1(A, delta(moveTo(P, H)))";
by (REPEAT (resolve_tac [allI] 1));
by (cut_facts_tac [Cm1'] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT(eresolve_tac [exE] 1));
by (REPEAT (resolve_tac [exI] 1));
by (SW_tac_asm(BSW,[Imp_refl],[l_ref,lemma13],1));
Qed "lemmaM2_2";

Goal "ALL A O P D . EX H .
\[ l(G,mobReq(A,O,P,D)) \land (G,toBeVetoed(mob(A,O,P),D)) \] BECAUSE
\[ l(A,delta(moveTo(P,H))) \]
by (REPEAT (resolve_tac [allI] 1));
by (cut_facts_tac [lemmaM2_2] 1);
by (REPEAT(dresolve_tac [spec] 1));
by (REPEAT(eresolve_tac [exE] 1));
by (cut_facts_tac [Gm1'] 1);
by (REPEAT(dresolve_tac [spec] 1));
by (REPEAT(eresolve_tac [exE] 1));
by (REPEAT (resolve_tac [exI] 1));
by (resolve_tac [BTR] 1);
by (SW_tac_asm(BSW,[Imp_refl],[l_ref,lemma13],1));
by (SW_tac_asm(BSW,[Imp_refl],[l_ref,lemma13],1));
Qed "lemmaM2_3";

Goal "ALL A O P D . EX H .
\[ l(G,mobReq(A,O,P,u)) \land (G,toBeVetoed(mob(A,O,P),D)) \] BECAUSE
\[ l(A,delta(moveTo(P,H))) \]
by (REPEAT (resolve_tac [allI] 1));
by (cut_facts_tac [lemmaM2_3] 1);
by (REPEAT(dresolve_tac [spec] 1));
by (REPEAT(eresolve_tac [exE] 1));
by (cut_facts_tac [Gm1'] 1);
by (REPEAT(dresolve_tac [spec] 1));
by (REPEAT(eresolve_tac [exE] 1));
by (REPEAT (resolve_tac [exI] 1));
by (SW_tac_asm(BSW,[lemma13],[Imp_refl],1));
Qed "lemmaM2_3bis";

Goal "ALL A T P D . EX H .
\[ l(T,mobReq(A,O,P,dest(T))) \] BECAUSE
\[ l(A,delta(moveTo(P,H))) \]
by (REPEAT (resolve_tac [allI] 1));
by (cut_facts_tac [Gm2'] 1);
by (REPEAT(dresolve_tac [spec] 1));
by (REPEAT(eresolve_tac [exE] 1));
by (cut_facts_tac [Gm1'] 1);
by (REPEAT(dresolve_tac [spec] 1));
by (REPEAT(eresolve_tac [exE] 1));
by (cut_facts_tac [Cm1'] 1);
by (REPEAT(dresolve_tac [spec] 1));
by (REPEAT(eresolve_tac [exE] 1));
by (REPEAT (resolve_tac [exI] 1));
by (resolve_tac [BTR] 1);
by (SW_tac_asm(BSW,[Imp_refl],[l_ref,lemma13],1));
by (resolve_tac [BTR] 1);
by (SW_tac_asm(BSW,[Imp_refl],[l_ref,lemma13],1));
by (SW_tac_asm(BSW,[Imp_refl],[l_ref,lemma13],1));
qed "lemmaM2_4";

Goal "ALL A T P D . EX H .
  \( l(T,\text{mobReq}(A, O, P, \text{dest}(T))) \& (T, \\text{toBeVetoed}(\text{mob}(A, O, P, D))) \)
  \( \because l(A, \text{delta}(\text{moveTo}(P, H))) \)"
by (REPEAT (resolve_tac [allI] 1));
by (cut_facts_tac [lemmaM2_4] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (REPEAT (resolve_tac [exI] 1));
by (SW_tac_asm (BSW, [lemma13], [Imp_ref1], [Imp_ref1]1));
qed "lemmaM2_4bis";

Goal "ALL G A O P St . EX H1 H2 H3 .
  \( l(G, \text{mobReq}(A, O, F, i)) \)
  \( l(G, \text{mobReq}(A, O, F, u)) \)
  \( l(T, \text{mobReq}(A, O, P, \text{dest}(T))) \because l(A, \text{delta}(\text{moveTo}(P, H1))) \)
  \( l(A, \text{delta}(\text{moveTo}(P, H2))) \)
  \( l(A, \text{delta}(\text{moveTo}(P, H3))) \)"
by (REPEAT (resolve_tac [allI] 1));
by (cut_facts_tac [lemmaM2_2] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [lemmaM2_3] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [lemmaM2_4] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (REPEAT (resolve_tac [exI] 1));
by (resolve_tac [BPD] 1);
by (SW_tac_asm (BSW, [Imp_ref1], [lemma21], [Imp_ref1], [Imp_ref1]1));
by (resolve_tac [BPD] 1);
by (SW_tac_asm (BSW, [Imp_ref1], [], [Imp_ref1], [Imp_ref1], [Imp_ref1]), 1));
by (SELECT_GOAL (Auto_tac) 1);
by (SW_tac_asm (BSW, [Imp_ref1], [], [Imp_ref1], [Imp_ref1]), 1));
by (SELECT_GOAL (Auto_tac) 1);
qed "lemmaM2_5";

Goal "ALL A O P D T G . EX H1 H2 H3 .
  \( l(0, \text{mobReq}(A, O, P, i)) \& l(0, \\text{toBeVetoed}(\text{mob}(A, O, P, D))) \)
  \( l(G, \text{mobReq}(A, O, P, u)) \& l(G, \\text{toBeVetoed}(\text{mob}(A, O, P, D))) \)
  \( l(T, \text{mobReq}(A, O, P, \text{dest}(T))) \& l(T, \\text{toBeVetoed}(\text{mob}(A, O, P, D))) \)
  \( \because l(A, \text{delta}(\text{moveTo}(P, H1))) \)
  \( l(A, \text{delta}(\text{moveTo}(P, H2))) \)
  \( l(A, \text{delta}(\text{moveTo}(P, H3))) \)"
by (REPEAT (resolve_tac [allI] 1));
by (cut_facts_tac [lemmaM2_5] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (REPEAT (resolve_tac [exI] 1));
by (SW_tac_asm (BSW, [], [Imp_ref1], [Imp_ref1], [Imp_ref1], 1));
by (Auto_tac);
qed "lemmaM2_5bis";

Goal "ALL A O P D T G . EX H .
  \( l(0, \text{mobReq}(A, O, P, i)) \)
  \( l(G, \text{mobReq}(A, O, P, u)) \)
  \( l(T, \text{mobReq}(A, O, P, \text{dest}(T))) \because l(A, \text{delta}(\text{moveTo}(P, H))) \)"
Goal "ALL A O P St . EX H.
\[ l(G, \text{mobReq}(A, O, P, St)) \text{ BECAUSE } l(A, \text{delta}(\text{moveTo}(P, H))) \]
by (REPEAT (resolve_tac [allI] 1));
by (cut_facts_tac [lemmaM2_5] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (REPEAT (TRY((eresolve_tac [BdisjE] 1)) THEN
(REPEAT (resolve_tac [exI] 1)) THEN
(assume_tac 1))));
qed "lemmaM2_6";

Goal "ALL A O P D T G . EX H.
\[ l(G, \text{mobReq}(A, O, P, i)) \& l(G, \text{toBeVetoed}(\text{mob}(A, O, P, D)) ||
l(T, \text{mobReq}(A, O, P, u)) \& l(T, \text{toBeVetoed}(\text{mob}(A, O, P, D))) \]
BECAUSE \[ l(A, \text{delta}(\text{moveTo}(P, H))) \]
by (REPEAT (resolve_tac [allI] 1));
by (cut_facts_tac [lemmaM2_6] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (REPEAT (resolve_tac [exI] 1));
by (SW_tac_asm(BSW, [], [Imp_refl], 1));
by (assume_tac 1);
qed "Id1";

Goal "ALL A O P D . EX H.
\[ l(A, \text{exc}(\text{mob}(A, O, P, D)) \text{ BECAUSE } l(A, \text{delta}(\text{moveTo}(P, H))) \]
by (REPEAT (resolve_tac [allI] 1));
by (cut_facts_tac [lemmaM2_1] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Gm9] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [lemmaM2_6bis] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (REPEAT (resolve_tac [exI] 1));
by (resolve_tac [BTR] 1);
by (SW_tac_asm(BSW, [Imp_refl], [], 1));
by (assume_tac 1);
by (assume_tac 1);
qed "lemmaM2_7";

Goal "ALL A O P D . EX G St H.
\[ l(A, \text{exc}(\text{mob}(A, O, P, D)) \text{ BECAUSE } l(A, \text{delta}(\text{moveTo}(P, H))) \&
l(G, \text{mobReq}(A, O, P, St)) \]&
l(G,toBeVetoed(mob(A,O,P),D))";
by (REPEAT (resolve_tac [allI] 1));
by (cut_facts_tac [lemmaM2_1] 1);
by (REPEAT(dresolve_tac [spec] 1));
by (REPEAT(eresolve_tac [exE] 1));
by (cut_facts_tac [lemmaM2_7] 1);
by (REPEAT(dresolve_tac [spec] 1));
by (REPEAT(eresolve_tac [exE] 1));
by (REPEAT (resolve_tac [exI] 1));
by (resolve_tac [BAnd] 1);
by (assume_tac 1);
by (assume_tac 1);
qed "M2";

Goal "ALL A . EX G . INIT(l(A,guardedby(G)))";
by (REPEAT (resolve_tac [allI] 1));
by (cut_facts_tac [ILA] 1);
by (REPEAT(dresolve_tac [spec] 1));
by (REPEAT(eresolve_tac [exE] 1));
by (REPEAT (resolve_tac [exI] 1));
by (resolve_tac [IN] 1);
by (assume_tac 1);
by (resolve_tac [lemma13] 1);
qed "lemmaMob1";

Goal "ALL A . EX G . INIT(l(A,guardedby(G) & history(nilCNL)))";
by (REPEAT (resolve_tac [allI] 1));
by (cut_facts_tac [lemmaMob1] 1);
by (REPEAT(dresolve_tac [spec] 1));
by (REPEAT(eresolve_tac [exE] 1));
by (cut_facts_tac [S4] 1);
by (REPEAT(dresolve_tac [spec] 1));
by (REPEAT(eresolve_tac [exE] 1));
by (REPEAT (resolve_tac [exI] 1));
by (resolve_tac [I3] 1);
by (resolve_tac [IN] 1);
by (resolve_tac [And_co_1] 2);
by (resolve_tac [IffSimpl] 2);
by (resolve_tac [IC] 1);
by (resolve_tac [DSTL.I2] 1);
by (assume_tac 1);
by (resolve_tac [DSTL.I2] 1);
by (assume_tac 1);
qed "lemmaMob2";

Goal "ALL A . co l(A, EX G . guardedby(G))";
by (REPEAT (resolve_tac [allI] 1));
by (cut_facts_tac [S5] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (resolve_tac [mp] 1);
by (assume_tac 2);
by (resolve_tac [mp] 1);
by (resolve_tac [K] 1);
by (resolve_tac [Nec] 1);
by (resolve_tac [impI] 1);
by (REPEAT (eresolve_tac [exE] 1));
by (resolve_tac [exI] 1);
by (resolve_tac [mp] 1);
by (assume_tac 2);
back();

199
by (resolve_tac [lemma13] 1); 
qed "lemmaMob3";

Goal "ALL G M S P R .
    1(G, delta(msgReq(M, S, P, rec(R)))) -->
    1(G, delta((msgReq(M, S, P, rec(R)))) & guarding(R) | guarding(R))";
by (REPEAT (resolve_tac [allI] 1));
by (resolve_tac [1_ref] 1);
by (Simp_tac 1);
qed "lemmaC1";

Goal "ALL S M P G . EX R D1 D2 D3 .
    1(S, delta(out(M, P)) & guardedby(G)) LEADSTO
    1(R, delta(in(M, S)));
    1(S, delta(exc(msg(M, S, P), D1)));
    1(S, delta(exc(msg(M, S, P), D2)));
    1(S, delta(exc(msg(M, S, P), D3)));
by (REPEAT (resolve_tac [allI] 1));
by (cut_facts_tac [Cc1] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Gc1] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Gc2] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Cc2] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Cc3] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Cc2] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Cc3] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Cc3] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (REPEAT (resolve_tac [exI] 1));
(* from the will to send a message of A to the communication request in O *)
by (Tran_tac(LTR, [], [], 1)); (*Cc1*)
by (assume_tac 1);
(* a guardian will receive the request and resolve the name of
 the receiver or the first guardian will vetoes the request *)
by (Tran_tac(LTR, [], [], 1)); (*Gc1*)
by (assume_tac 1);
by (resolve_tac [lpd] 1);
(* the guardian that resolves the address either guards the
 receiver or forward the request to a guardian that's
 guarding the receiver*)
by (SW_tac(LSW, [], [], [Imp_ref1], 1));
by (cut_facts_tac [lemmaCI] 1);
by (REPEAT (dresolve_tac [spec] 1));

200
by (REPEAT (eresolve_tac [exE] 1));
by (assume_tac 1);
by (SELECT_GOAL (rewrite_goals_tac [AndOr_distr]) 1);
by (SW_tac(LSW,[IffE_right,Or_\_I,IffSimm],[],[Imp_refl],1));
by (resolve_tac [LPD] 1);
(* the guardian that resolves the address guards the receiver and either delivers the message or vetoes it *)
by (Tran_tac(LTR,[LI],[],1));(*Cc2*);
by (assume_tac 1);
by (resolve_tac [LPD] 1);
(* the guardians that resolves the address delivers the message *)
by (SW_tac(LSW,[Imp_refl],[LI,LcI],[],1));
by (SELECT_GOAL (Auto_tac) 1);
(* the guardians that resolves the address vetoes the request *)
by (SW_tac_asm(LSW,[Imp_refl],[],1));(*Cc3*);
by (SELECT_GOAL (Auto_tac) 1);
(* the guardian that resolves the address does not guards the receiver and forwards the message to a guardian that's guarding the receiver*)
by (Tran_tac(LTR,[],[],1));(*Gc2*);
by (assume_tac 1);
(* the guardians that's guarding the receiver either delivers the message or vetoes it *)
by (Tran_tac(LTR,[LI],[],1));(*Cc2*);
by (assume_tac 1);
by (resolve_tac [LPD] 1);
(* the guardians that's guarding the receiver delivers the message *)
by (SW_tac(LSW,[Imp_refl],[LI,LcI],[],1));
by (SELECT_GOAL (Auto_tac) 1);
(* the guardians that's guarding the receiver vetoes the request *)
by (SW_tac_asm(LSW,[Imp_refl],[],1));(*Cc3*);
by (SELECT_GOAL (Auto_tac) 1);
(* the first guardians vetoes the request *)
by (SW_tac_asm(LSW,[Imp_refl],[],1));(*Cc3*);
by (SELECT_GOAL (Auto_tac) 1);qed "lemmaC2";

Goal "ALL S M P G . EX R D .
  \[ (\forall S . \delta(out(M,P))\& guardingby(G)) \leadsto \\
  \[ (\forall R \forall \delta(in(M,S)))\& \delta(exc(msg(M,S,P,D)))\]
";
by (REPEAT(resolve_tac [allI] 1));
by (cut_facts_tac [lemmaC2] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (REPEAT ((TRY(eresolve_tac [LdisjE] 1) THEN 
(TRY (resolve_tac [exI] 1)) THEN 
(SW_tac_asm(LSW,[Imp_refl],[],1)) THEN 
(SELECT_GOAL (Auto_tac) 1)))));
qed "C";

Goal "ALL R M S . EX P G St .
  \[ (\forall R .\in(M,S)) \because \\
  \[ (\forall G \delta(msgReq(M,S,P,St))\& satisfy(consCNL(R,nilCNL),P))\]
";
by (REPEAT(resolve_tac [allI] 1));
by (cut_facts_tac [Cc2'] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT(eresolve_tac [exE] 1));
by (cut_facts_tac [Gcl_2'] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Ccl_1'] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Ccl_2'] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (resolve_tac [BTR] 1);
by (assume_tac 1);
by (SW_tac(BSW,[l_ref,lemma13],[],[Imp_refl],1));
by (resolve_tac [BTR] 1);
by (assume_tac 1);
by (resolve_tac [BAnd]1);
by (resolve_tac [B1] 2);
by (resolve_tac [BcI] 2);
by (resolve_tac [BTR] 1);
by (SW_tac(BSW,[l_ref,lemma13],[],[Imp_refl],1));
by (assume_tac 1);
by (SW_tac_asm(BSW,[Imp_refl],[l_ref,lemma13],1));
qed "C1";

Goal "ALL R M S . EX G1 P St1 G2 St2 .
  l(R,in(M,S)) BECAUSE
  l(G1,delta(msgReq(M,S,P,St1))&guarding(R))&
  l(G2,delta(msgReq(M,S,P,St2))&guarding(S))";
by (REPEAT(resolve_tac [allI] 1));
by (cut_facts_tac [Cc2'] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Cc2'] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Gcl_2'] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Ccl_1'] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (REPEAT (resolve_tac [exI] 1));
by (resolve_tac [BCC] 1);
(* from the receipt of a message in A to the receipt of the
communication request for A in the guardian guarding A*)
by (assume_tac 1);(*Cc2*)
(* from the receipt of a message in A to the receipt of the
communication request still unresolved in the guardian
guarding the sender*)
by (Tran_tac(BTR,[],[],1));
by (assume_tac 1);(*Cc2*)
by (Tran_tac(BTR,[],[],1));
by (SW_tac_asm(BSW,[l_ref,lemma13],[Imp_refl],1));(*Gcl_2*)
by (SW_tac_asm(BSW,[l_ref,lemma13],[Imp_refl],1));
qed "A2";

Goal "ALL G M S P .
  l(G,delta(msgReq(M,S,P,u))) BECAUSE l(S,delta(out(M,P)))";
by (REPEAT(resolve_tac [allI] 1));
by (cut_facts_tac [Ccl_1'] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT(eresolve_tac [exE] 1));
by (cut_facts_tac [Ccl_2'] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT(eresolve_tac [exE] 1));
by (REPEAT(resolve_tac [exI] 1));
by (resolve_tac [BTR] 1);
by (assume_tac 1);
by (SW_tac_asm(BSW,[Imp_ref1],[l_ref,lemma13],1));
qed "lemmaC2_1";

Goal "ALL G M S P .
  l(G,delta(msgReq(M,S,P,rec(R)))) BECAUSE
  l(S,delta(out(M,P)))";
by (REPEAT(resolve_tac [allI] 1));
by (cut_facts_tac [Gcl_2'] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT(eresolve_tac [exE] 1));
by (cut_facts_tac [lemmaC2_1] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT(eresolve_tac [exE] 1));
by (REPEAT(resolve_tac [exI] 1));
by (resolve_tac [BTR] 1);
by (SW_tac_asm(BSW,[Imp_ref1],[l_ref,lemma13],1));
by (assume_tac 1);
qed "lemmaC2_2";

Goal "ALL G M S P R .
  l(G,delta(msgReq(M,S,P,rec(R))))|l(G,delta(msgReq(M,S,P,rec(R))))
    BECAUSE l(S,delta(out(M,P)))";
by (REPEAT(resolve_tac [allI] 1));
by (cut_facts_tac [lemmaC2_1] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT(eresolve_tac [exE] 1));
by (cut_facts_tac [lemmaC2_2] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT(eresolve_tac [exE] 1));
by (REPEAT(resolve_tac [exI] 1));
by (resolve_tac [BPD] 1);
by (assume_tac 1);
by (assume_tac 1);
qed "lemmaC2_3";

Goal "ALL G M S P St .
  l(G,delta(msgReq(M,S,P,St))) BECAUSE
  l(S,delta(out(M,P)))";
by (REPEAT(resolve_tac [allI] 1));
by (cut_facts_tac [Gc3] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT(eresolve_tac [exE] 1));
by (cut_facts_tac [lemmaC2_3] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT(eresolve_tac [exE] 1));
by (REPEAT(resolve_tac [exI] 1));
by (SW_tac_asm(BSW,[],[Imp_ref1],1));
by (assume_tac 1);
qed "Id2";

Goal "ALL S M P D . EX G St .
  l(S,exc(msg(M,S,P),D)) BECAUSE
l(S,delta(out(M,P))) & 
l(G,delta(msgReq(M,S,P,St))) & 
l(G,tcBeVetoed(msg(M,S,P),D))";
by (REPEAT(resolve_tac [allI] 1));
by (cut_facts_tac [Cc3'] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT(eresolve_tac [exE] 1));
by (cut_facts_tac [Gc1_2'] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT(eresolve_tac [exE] 1));
by (cut_facts_tac [Id2] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT(eresolve_tac [exE] 1));
by (cut_facts_tac [] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT(eresolve_tac [exE] 1));
by (REPEAT(resolve_tac [exI] 1));
by (RESOLVE_TAC [BTR] 1);
by (assume_tac 1);
by (resolve_tac [BAnd] 1);
by (resolve_tac [BAnd] 2);
by (assume_tac 2);
by (resolve_tac [BI] 2);
by (resolve_tac [BcI] 2);
by (resolve_tac [BTR] 1);
by (assume_tac 1);
by (assume_tac 1);
qed "C2";
The theories of the examples

mob_addl_prof_name_refModel.ML

Goal "ALL A T H O . EX D1 D2 D3 .
l(A, delta(moveTo(consP(name(T),nilP),H)) & guardedby(O)) LEADSTO
l(A, delta(guardedby(T)))|
  delta(exc(mob(A,0,consP(name(T),nilP)),D1))
  delta(exc(mob(A,0,consP(name(T),nilP)),D2))
  delta(exc(mob(A,0,consP(name(T),nilP)),D3))
delta(exc(mob(A,0,consP(name(T),nilP)),dr)))";
by (REPEAT(resolve_tac [allI] 1));
by (cut_facts_tac [Gml_name] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Gm2_name] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Gm3] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Gm5] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Gm2] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Gm3] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Gm3] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Gm3] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Gm3] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Gm3] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Gm3] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (REPEAT (resolve_tac [LTR] 1));
by (resolve_tac [LTB1] 1);
by (resolve_tac [U2S] 4);
by (cut_facts_tac [A3] 4);
by (REPEAT (dresolve_tac [spec] 4));
by (REPEAT (eresolve_tac [exE] 4));
by (assume_tac 4);
by (cut_facts_tac [A4] 4);
by (REPEAT (dresolve_tac [spec] 4));
by (REPEAT (eresolve_tac [exE] 4));
by (assume_tac 4);
by (cut_facts_tac [Cml] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (assume_tac 1);
by (SW_tac(BSW,[l_ref,lemma14],[],[Imp_refl],1));
by (cut_facts_tac[Cm2'] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (assume_tac 1);
by (resolve_tac [l_ref] 1);
by (SELECT_GOAL (Auto_tac) 1);
(* from the mobility request in 0 to the outcome of the request *)
(* splitting the cases on the condition (guarding(A)|moving(A)) *)
by (SELECT_GOAL (rewrite_goals_tac [AndOr_distr]) 1);
by (SW_tac(LSW,[IffE_right,Or_|,IffSmm],[],[],[Imp_refl],1));
by (resolve_tac [LPD] 1);
(* the mobility request reaches the guardian when guarding(A) holds *)
(* the guardian starts serving the request, either it forwards it to another guardian or vetoes it *)
by (Tran_tac(LTR,[],[],1));
by (assume_tac 1); (**)
by (resolve_tac [LPD] 1);
(* the guardian forwards the request *)
(* the destination guardian receives the request or another guardian vetoes it *)
(* by (Tran_tac(LTR,[Gm2],[],1));*)
by (Tran_tac(LTR,[],[],1));
by (assume_tac 1); (**)
by (resolve_tac [LPD] 1);
(* the destination guardian receives the request and either accepts or vetoes it *)
by (Tran_tac(LTR,[],[],1));
by (assume_tac 1); (**)
by (resolve_tac [LPD] 1);
(* the destination guardian accepts the request *)
by (SW_tac_asm(LSW,[Imp_refl],[l_ref],1));
by (SELECT_GOAL (Auto_tac) 1);
(* the destination guardian vetoes the request *)
by (SW_tac_asm(LSW,[Imp_refl],[l_ref],1));
by (SELECT_GOAL (Auto_tac) 1);
(* another guardian vetoes the request *)
by (SW_tac_asm(LSW,[Imp_refl],[l_ref],1));
by (SELECT_GOAL (Auto_tac) 1);
(* the guardian vetoes the requests *)
by (SW_tac_asm(LSW,[Imp_refl],[l_ref],1));
by (SELECT_GOAL (Auto_tac) 1);
(* the mobility request reaches the guardian and moving(A) holds *)
by (Tran_tac(LTR,[],[],1));
by (assume_tac 1);
by (SW_tac_asm(LSW,[Imp_refl],[l_ref],1));
by (SELECT_GOAL (Auto_tac) 1);
ged "lemmaM1_name";

Goal "ALL A T H O . EX D1 D2 D3.
1(A, delta(moveTo(consP(name(T),nilP),H))&guardedby(O)) LEADSTO
1(A, delta(guardedby(T)))
1(A, delta(exc(mob(A, consP(name(T),nilP)),D1)))
1(A, delta(exc(mob(A, consP(name(T),nilP)),D2)))
1(A, delta(exc(mob(A, consP(name(T),nilP)),D3)))"
l(A, delta(exc(mob(A,O,consP(name(T),nilP)),dr)))";
by (REPEAT(resolve_tac [all1] 1));
by (cut_facts_tac [lemmaM1_name] 1);
by (REPEAT(dresolve_tac [spec] 1));
by (REPEAT(eresolve_tac [exE] 1));
by (REPEAT(resolve_tac [exI] 1));
by (SW_tac_asm(LSW,[Imp_refl],[l],1));
by (resolve_tac [IffE_right] 1);
by (SELECT_GOAL (rewrite_goals_tac [Or_l_eq]) 1);
by (Simp_tac 1);
qed "lemmaM2_name";

Goal "ALL A T H O . EX D .
l(A,delta(moveTo(consP(name(T),nilP),H))&guardedby(O))
  LEADSTO l(A,delta(guardedby(T))|d)
      delta(exc(mob(A,O,consP(name(T),nilP)),D))";
by (REPEAT(resolve_tac [all1] 1));
by (cut_facts_tac [lemmaM2_name] 1);
by (REPEAT(dresolve_tac [spec] 1));
by (REPEAT(eresolve_tac [exE] 1));
by (REPEAT(TRY(eresolve_tac [LdisjE] 1) THEN
                     (REPEAT (resolve_tac [exI] 1)) THEN
                     (SW_tac_asm(LSW,[Imp_refl],[l],1)) THEN
                     (SELECT_GOAL (Auto_tac 1))));
qed "M_name";

Goal "ALL S M R G . EX D1 D2 D3 .
l(S,delta(out(M,consP(name(R),nilP)))&guardedby(G)) LEADSTO
 l(R,delta(in(M,S))|d)
  l(S,delta(exc(msg(M,S,consP(name(R),nilP)),D1)))|d)
 l(S,delta(exc(msg(M,S,consP(name(R),nilP)),D2))|d)
  l(S,delta(exc(msg(M,S,consP(name(R),nilP)),D3))|d))";
by (REPEAT (resolve_tac [all1] 1));
by (cut_facts_tac [Cc1] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Gc1_name] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Gc2] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Cc2] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Cc3] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Gc2] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Cc2] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Cc3] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Cc3] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Cc3] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (cut_facts_tac [Cc3] 1);
(* from the will to send a message of A to the communication request in O *)
by (Tran_tac(LTR, [[],[]],1));(*Cc1*)
by (assume_tac 1);
(* a guardian will receive the request and resolve the name of the receiver or the first guardian will vetoes the request *)
by (Tran_tac(LTR, [[],[]],1));(*Gc1*)
by (assume_tac 1);
by (resolve_tac [LPD] 1);
(* the guardian that resolves the address either guards the receiver or forward the request to a guardian that's guarding the receiver*)
by (SW_tac(LSW, [[],[],[Imp_refl],1]));
by (cut_facts_tac [lemmaC1] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (assume_tac 1);
by (SELECT_GOAL (rewrite_goals_tac [AndOr_distr]) 1);
by (SW_tac(LSW, [Iff-right,Or-Introduced,mp],[],[Imp_refl],1));
by (resolve_tac [LPD] 1);
(* the guardian that resolves the address guards the receiver and either delivers the message or vetoes it *)
by (Tran_tac(LTR, [LI],[],1));(*Cc2*)
by (assume_tac 1);
by (resolve_tac [LPD] 1);
(* the guardians that resolves the address delivers the message *)
by (SW_tac(LSW, [Imp_refl],[],[LI,LcI],[],1));
by (SELECT_GOAL (Auto_tac) 1);
(* the guardians that resolves the address vetoes the request*)
by (SW_tac_asm(LSW, [Imp_refl],[],1));(*Cc3*)
by (SELECT_GOAL (Auto_tac) 1);
(* the guardian that resolves the address does not guards the receiver and forwards the message to a guardian that's guarding the receiver*)
by (Tran_tac(LTR, [[],[]],1));(*Gc2*)
by (assume_tac 1);
(* the guardians that's guarding the receiver either delivers the message or vetoes it *)
by (Tran_tac(LTR, [LI],[],1));(*Cc2*)
by (assume_tac 1);
by (resolve_tac [LPD] 1);
(* the guardians that's guarding the receiver delivers the message *)
by (SW_tac(LSW, [Imp_refl],[],[LI,LcI],[],1));
by (SELECT_GOAL (Auto_tac) 1);
(* the guardians that's guarding the receiver vetoes the request *)
by (SW_tac_asm(LSW, [Imp_refl],[],1));(*Cc3*)
by (SELECT_GOAL (Auto_tac) 1);
(* the first guardians vetoes the request *)
by (SW_tac_asm(LSW, [Imp_refl],[],1));(*Cc3*)
by (SELECT_GOAL (Auto_tac) 1);
qed "lemmaC2_name";

Goal "ALL S M R G . EX D .
  1(S,delta(out(M,consP(name(R),nilP))))&guardedby(G))
LEADSTO
  1(R,delta(in(M,S)))) |
  1(S,delta(exc(msg(M,S,consP(name(R),nilP)),D))))";
by (REPEAT(resolve_tac [allI] 1));
by (cut_facts_tac [lemmaC2_name] 1);
by (REPEAT (dresolve_tac [spec] 1));
by (REPEAT (eresolve_tac [exE] 1));
by (REPEAT (TRY (eresolve_tac [LdisjE] 1) THEN
    (REPEAT (resolve_tac [exI] 1)) THEN
    (SW_tac_asm (LSW,[Imp_ref1],[],1)) THEN
    (SELECT_GOAL (Auto_tac 1))));
qed "C_name";
mob_adtl_services_refModel.ML

(* service discovery reference model *)

Goal "ALL A F P G . EX S .
  1(A, delta(service_req(F,P))&guardedby(G)) LEADSTO
  1(A, service_available(S,F,P)|no_service_available(F,P))";
by (REPEAT(resolve_tac [allI] 1));
by (cut_facts_tac [Ser2_1] 1);
by (REPEAT(dresolve_tac [spec] 1));
by (REPEAT(eresolve_tac [exe] 1));
by (cut_facts_tac [Ser2_5] 1);
by (REPEAT(dresolve_tac [spec] 1));
by (REPEAT(eresolve_tac [exe] 1));
by (cut_facts_tac [Ser2_2] 1);
by (REPEAT(dresolve_tac [spec] 1));
by (REPEAT(eresolve_tac [exe] 1));
by (REPEAT(resolve_tac [spec] 1));
by (REPEAT(eresolve_tac [exe] 1));
by (assume_tac 1);
by (resolve_tac [LTR] 1);
by (assume_tac 1);
by (resolve_tac [LSW] 1);
by (resolve_tac [IffE_right] 1);
by (resolve_tac [Or_1] 1);
by (simp_tac 1);
by (resolve_tac [Imp_ref] 2);
by (resolve_tac [LPD] 1);
by (SW_tac_asm(LSW,[Imp_ref],[l_ref,lemma21],1));
by (SW_tac_asm(LSW,[Imp_ref],[l_ref,lemma22],1));
ged "Ser1_3";

Goal "ALL A S F P . EX G .
  1(A, service_available(S,F,P)|no_service_available(F,P)) BECAUSE
  1(A, delta(service_req(F,P))&guardedby(G))";
by (REPEAT(resolve_tac [allI] 1));
by (cut_facts_tac [Ser2_2_3'] 1);
by (REPEAT(dresolve_tac [spec] 1));
by (REPEAT(eresolve_tac [exe] 1));
by (cut_facts_tac [Ser2_1'] 1);
by (REPEAT(dresolve_tac [spec] 1));
by (REPEAT(eresolve_tac [exe] 1));
by (REPEAT(resolve_tac [exe] 1));
by (resolve_tac [BTR] 1);
by (assume_tac 1);
by (assume_tac 1);
ged "Ser1_3'";

Goal "ALL S F P . EX G .
  1(A, service_available(S,F,P)) BECAUSE
  1(G, satisfy(consCNL(S,nilCNL),
    consP(service_profile(F,P,A),nilP)))";
by (REPEAT(resolve_tac [allI] 1));
by (cut_facts_tac [Ser2_2'] 1);
by (REPEAT(dresolve_tac [spec] 1));
by (REPEAT(eresolve_tac [exe] 1));
by (cut_facts_tac [Ser2_6] 1);
by (REPEAT(dresolve_tac [spec] 1));
by (REPEAT(eresolve_tac [exE] 1));
by (REPEAT(resolve_tac [exI] 1));
by (SW_tac_asm(BSW,[Imp_refl],[],1));
by (resolve_tac [mp] 1);
by (resolve_tac [D3] 1);
by (Simp_tac 1);
by (assume_tac 1);
qed "Ser1_3'";

Goal "ALL A F P . EX G .
   1(A,no_service_available(F,P)) BECAUSE
     1(G,unsatisfiable(consP(service_profile(F,P,A),nilP)))";
by (REPEAT(resolve_tac [allI] 1));
by (cut_facts_tac [Ser2_3'] 1);
by (REPEAT(dresolve_tac [spec] 1));
by (REPEAT(eresolve_tac [exE] 1));
by (cut_facts_tac [Ser2_7] 1);
by (REPEAT(dresolve_tac [spec] 1));
by (REPEAT(eresolve_tac [exE] 1));
by (REPEAT(resolve_tac [exI] 1));
by (SW_tac_asm(BSW,[Imp_refl],[],1));
by (resolve_tac [mp] 1);
by (resolve_tac [D3] 1);
by (Simp_tac 1);
by (assume_tac 1);
qed "Ser1_3'''";
mob_adtl_migAgents.ML

(* properties of the reference model for the migrating agents case study *)

Goal "ALL G A . EX T H.
  1(G,delta(wantToLeave)&guarding(A)) LEADSTO
  1(G,satisfy(conscNL(T,nilCNL),consP(pcMeeting,nilP)))&
  1(A,delta(moveTo(consP(name(T),nilP),H))&guardedby(G)))";
by (REPEAT(resolve_tac [allI] 1));
by (cut_facts_tac [Gpc1] 1);
by (REPEAT(dresolve_tac [spec] 1));
by (REPEAT(eresolve_tac [exE] 1));
by (cut_facts_tac [Gpc1] 1);
by (REPEAT(dresolve_tac [spec] 1));
by (REPEAT(eresolve_tac [exE] 1));
by (cut_facts_tac [Gpc8] 1);
by (REPEAT(dresolve_tac [spec] 1));
by (REPEAT(eresolve_tac [exE] 1));
by (REPEAT(resolve_tac [exI] 1));
by (resolve_tac [LTR] 1);
by (assume_tac 1);
by (resolve_tac [LCC] 1);
by (SW_tac(LSW,[lemma14],[[]],[Imp_refl],1));
by (SW_tac(LSW,[1_ref,lemma13],[LI,LcI],[mp,D3],1));
by (Simp_tac 1);
by (assume_tac 1);
by (SW_tac_asm(LSW,[lemma14],[Imp_refl],1));
ged "migAgents_lemma1";

Goal "ALL G A . EX T O D.
  1(G,delta(wantToLeave)&guarding(A)) LEADSTO
  1(G,satisfy(conscNL(T,nilCNL),consP(pcMeeting,nilP)))&
  1(A,delta(exc(mob(A,O,consP(name(T),nilP)),D)))";
by (REPEAT(resolve_tac [allI] 1));
by (cut_facts_tac [PCmeeting_lemma1] 1);
by (REPEAT(dresolve_tac [spec] 1));
by (REPEAT(eresolve_tac [exE] 1));
by (cut_facts_tac [M_name] 1);
by (REPEAT(dresolve_tac [spec] 1));
by (REPEAT(eresolve_tac [exE] 1));
by (REPEAT(resolve_tac [exI] 1));
by (resolve_tac [LTR] 1);
by (assume_tac 1);
by (resolve_tac [LCC] 1);
by (SW_tac(LSW,[lemma13],[LI,LcI],[Imp_refl],1));
by (SW_tac_asm(LSW,[lemma14],[Imp_refl],1));
ged "migAgents_lemma2";

Goal "ALL A O T D . co_l(A,exc(mob(A,O,consP(name(T),nilP)),D))";
by (REPEAT(resolve_tac [allI] 1));
by (cut_facts_tac [Cm3'] 1);
by (REPEAT(dresolve_tac [spec] 1));
by (REPEAT(eresolve_tac [exE] 1));
by (cut_facts_tac [Gpc9] 1);
by (REPEAT(dresolve_tac [spec] 1));
by (REPEAT(eresolve_tac [exE] 1));
by (cut_facts_tac [Ta] 1);
by (REPEAT(dresolve_tac [spec] 1));
by (REPEAT(eresolve_tac [exE] 1));
by (REPEAT(resolve_tac [exI] 1));
by (resolve_tac [co_l_notI] 1);
by (assume_tac 1);
by (assume_tac 1);
by (resolve_tac [IW] 1);
by (assume_tac 1);
by (resolve_tac [i_ref] 1);
by (Auto_tac);
qed "migAgents_lemma3";

Goal "ALL A O T D .
co_l(A,~delta(exc(mob(A,O,consP(name(T),nilP)),D)))";
by (REPEAT(resolve_tac [allI] 1));
by (cut_facts_tac [PCMeeting_lemma3] 1);
by (REPEAT(dresolve_tac [spec] 1));
by (REPEAT(eresolve_tac [exE] 1));
by (REPEAT(resolve_tac [exI] 1));
by (resolve_tac [mp] 1);
by (assume_tac 2);
by (resolve_tac [mp] 1);
by (resolve_tac [K] 1);
by (resolve_tac [Nec] 1);
by (resolve_tac [ContrapositionPN] 1);
by (resolve_tac [deltaE] 1);
qed "migAgents_lemma4";

Goal "ALL G A . EX T .
1(G,delta(wantToLeave)&guarding(A)) LEADsto
1(G,satisfy(consCNL(T,nilCNL),consP(pcMeeting,nilP))) &
1(A,delta(guardedBy(T)))";
by (REPEAT(resolve_tac [allI] 1));
by (cut_facts_tac [PCMeeting_lemma2] 1);
by (REPEAT(dresolve_tac [spec] 1));
by (REPEAT(eresolve_tac [exE] 1));
by (REPEAT(eresolve_tac [exI] 1));
by (REPEAT(resolve_tac [exI] 1));
by (SW_tac_asm(LSW,[Imp_ref1],[],1));
by (Auto_tac);
by (resolve_tac [impE] 1);
by (assume_tac 3);
by (assume_tac 2);
back();
by (resolve_tac [mp] 1);
by (resolve_tac [D3] 1);
by (Simp_tac 1);
by (resolve_tac [mp] 1);
by (assume_tac 2);
back();
back();
back();
by (resolve_tac [mp] 1);
by (resolve_tac [K] 1);
by (resolve_tac [Nec] 1);
by (Auto_tac);
qed "migAgents_prop";