Defining Inheritance in *i** at the Level of SR Intentional Elements[†]

Lidia López, Xavier Franch, Jordi Marco

Universitat Politècnica de Catalunya, C/Jordi Girona, 1-3, 08034 Barcelona, Spain
{llopez, franch, jmarco}@lsi.upc.edu

Abstract. The is-a relationship among actors has been introduced since the very beginning in the *i** framework. However, the effect of this construct at the level of intentional elements and dependencies is not always completely determined. In this paper, we explore the semantics of inheritance in *i** with focus on SR models. Aligning with its usual meaning in object-orientation, we distinguish 3 main notions to be defined: extension, refinement, and redefinition. For each of them, we have studied its effects on the different types of intentional elements and their links, and also dependencies, making explicit what can be and cannot be done. We have also analysed the proposal with an example that makes intensive use of inheritance, a multi-stakeholder distributed system in which different types of related stakeholders co-exist.

1 Introduction

Several variations of the *i** framework exist, for instance Yu's seminal proposal, GRL, and Tropos. They all agree on a core of main concepts whilst not addressing in much detail other related concepts (see [1] for an analysis). One of the elements whose definition is not complete is the concept of inheritance, despite the fact that it was incorporated in the framework since its early definition. Several authors make use of inheritance but they have not clearly defined this concept nor provided guidelines for usage. The reason for this lack of rigor in inheritance definition is that the construct is not needed often for some modeling tasks, and when needed, normally it just suffices with establishing inheritance of actors at the level of SD diagrams. But there are domains that need a more precise definition of inheritance.

As one of these domains, we have started to use the *i** language to model service-oriented multi-stakeholder distributed systems (MSDS). MSDS are distributed systems in which subsets of the nodes are designed, owned, or operated by distinct stakeholders. Using basic *i** modeling concepts such as intentional elements, links, and actors we experienced some limitations of *i** when specifying the needs of heterogeneous stakeholders in a particular example of system, a web-based travel agency [2]. A significant problem we faced when modeling this MSDS was caused by the need to use inheritance for building hierarchies of actors without knowing accurately the consequences on their rationale of doing so. Specifically, when

[†] This work has been supported by the Spanish research project TIN2007-64753.

modeling our MSDS system, we aimed to model a common rationale in the superactor and a specific rationale in the subactors. Using inheritance as defined by Yu, we felt the need to determine which model transformation operations are valid and which are their implications in the context of specialization of actors.

This paper reports our results in the definition of inheritance in *i** that were presented in the AOIS'07 workshop [3]. Also we have discussed the applicability of inheritance in the MSDS domain, reported in the VaMoS'08 workshop [4].

2 Objectives of the Research

The main objective of this research is: Presenting a complete and non-ambiguous definition of inheritance for the i* framework. This general goal may be split into:

- Studying the meaning of inheritance in the *i** framework. We are interested in exploring in which part of *i** models, and under which conditions, inheritance may be applied. Also, how inheritance affects subactor goals and dependencies. How the subactor goals can be modified to achieve these new dependencies or if it is possible that this modified behaviour can create new outgoing dependencies.
- Proposing a way to model inheritance in the *i** framework. As important as to define inheritance is how to model it. *i** makes intensive use of graphical elements to express actors' goals. So, we need an easy way to model inheritance. Of course, tool support is an important issue so that inheritance can be useful.
- Exploring how does inheritance affect to *i** treatments and properties. Treatments (e.g., backward reasoning) and properties (e.g., workability) are used to analyse models, so it is important that models that use inheritance can be also analysed.

We aim at validating the inheritance definition both formally and methodologically. For formal validation, we mean verifying that the proposal is sound and complete. For methodological validation, we mean to find out if *i** users like it, knowing if they would use it in their models. We are interested in:

- Knowing if the proposal is easy to learn. For this validation, we will ask some non-expert *i** users to use inheritance to model some academic examples.
- Knowing if our proposal is useful. For this property validation, we will present our proposal to *i** community using scientific events and the other means (e.g., the *i** wiki). We aim at applying this proposal to huge examples and even real projects.

3 Scientific Contributions

A goal of our proposal is to align *i** inheritance with the general concept of inheritance as known in OO approaches. After an analysis of existing options, we have decided to adhere to Meyer's *Taxomania rule*: "Every heir must introduce a feature, redeclare an inherited feature, or add an invariant clause". Upon adopting this rule in the *i** framework we obtain three different specialization operations on IE: extension (i.e., introducing a feature), redefinition (i.e., redeclaring an inherited feature), and refinement (i.e., adding an invariant clause):

- ***Extension***. In the OO paradigm, one of the most frequent ways of specializing a class is adding some information such as attributes and methods to a subclass. We extrapolate this idea into the *i** modeling framework and call it extension. Extension in *i** means adding new IEs to the SR model of a subactor together with relationships to other IEs. Any kind of IE, together with links and dependencies, may be added to the SR model of a subactor.
- ***Redefinition***. Redefinition ("redeclaration" in the Taxomania rule) allows redefining IEs and their relationships. The main difference among redefinition and refinement is that redefinition does not allow changing satisfactibility predicates. In the case of goals, tasks and resources, redefinition implies that the redefined IE (in the superclass) needs some decomposition using task-decomposition or means-ends links to make sense (otherwise, we would use extension or refinement). In the case of softgoals it is only possible to redefine the interpretation of the condition to be fulfilled (fit criterion), since they are not decomposed but just contributed.
- ***Refinement***. Refinement captures the third situation stated in Meyer's Taxomania rule, adding an invariant clause. We interpret adding an invariant as restricting the satisfactibility predicate of the IE being refined, in other words, satisfactibility of the new IE implies satisfactibility of the refined IE. More specifically, this means for the four types of IEs: (1) goals and (2) softgoals: the set of states attained by the new IE is a subset of the states attained in the refined IE; (3) tasks: the procedure to be undertaken in the new IE is more prescriptive than the procedure to be undertaken in the refined IE; (4) resource: the entity represented by the new IE entails more information than the entity represented by the refined IE.

As a result, specialization of an actor consists of several specialization operations applied to the inherited SR diagram. Extensions, refinements and redefinitions cannot not be arbitrary; conditions of applicability are explored in detail in [3].

Two important things that play a fundamental role in our approach are:

- ***Satisfactibilty***. Intuitively, an IE states some objective that may be satisfied or not. We assume that satisfactibility is denoted by a Boolean predicate. The exact meaning of satisfactibility depends on the type of the IE [3]. In extension and redefinition, the satisfactibility predicate does not change. However, by refinement, the satisfactibility predicate of an inherited IE is changed but not arbitrarily.
- ***Syntax***. The *i** framework heavily relies on the use of a graphical representation of its constructs. We want to apply a economy rule: Non-modified inherited IEs will not be included in the subactor unless strictly necessary. On the other hand, new IEs and modified inherited IEs will be included in the subactor SR using a solid line shape using the standard notation. When needed, non-modified inherited IEs will be included in the subactor SR using a dotted line shape; this is the only change in the standard use of *i** in our approach.

4 Conclusions

We have presented our first results towards defining in detail the concept of inheritance in the *i** framework. The main strengths of our approach are:

- It relies on the theory of inheritance as defined by some milestone references. Thus, it is compliant with the most recognised principles in this context.
- We avoided adding new constructs to *i**. This is an important issue since we avoid committing our approach to a particular version of the language. The only exception is in syntax (dotted lines for representing replicated elements).
- We have analyzed the effects of the several specialization constructs to the diversity of intentional elements, links and dependencies that are in *i** definition.

This work has been developed in the context of a collaboration with the Johannes Kepler University at Linz, Austria. In this context, the use of inheritance as a way of identifying candidate variation points in variability modelling is reported in [4].

5 Ongoing and future work

Our future work includes formalisation and the addition of inheritance into the *i** metamodel [1]. We will also focus on the specialization of dependencies in SD models and the transitivity of actor specialization. Another research question is to investigate the joint application of redefinition and refinement.

We are currently addressing these challenges also including research on adequate tool support for *i** inheritance. We are currently extending the model edition part of the J-PR*i*M tool [5] for supporting the inheritance concept, including exportation using iStarML [6].

References

1. Ayala C., Cares C., Carvallo J.P., Grau G., Haya M., Salazar G., Franch X., Mayol E. and Quer C.: A Comparative Analysis of *i**-Based Agent-Oriented Modeling Languages. *Proceedings of the 17th International Conference on Software Engineering and Knowledge Engineering (SEKE'05)*, 2005
2. Clotet R., Franch X., Grünbacher P., López L., Marco J., Quintus M., Seyff N.: Requirements Modelling for Multi-Stakeholder Distributed Systems: Challenges and Techniques. *Proceedings of the 1st International Conference on Research Challenges for Information Systems (RCIS'07)*, 2007
3. Clotet R., Franch X., López L., Marco J., Seyff N. and Grünbacher P.: The Meaning of Inheritance in *i**. *Proceedings of the 17th International Workshop on Agent-Oriented Information Systems (AOIS'07)*, 2007
4. Clotet R., Dhungana D., Franch X., Grünbacher P., López L., Marco J., Seyff, N.: Dealing with Changes in Service-Oriented Computing Through Integrated Goal and Variability Modeling. *Proceedings of the 2nd Workshop on Variability Modelling of Software-Intensive Systems (VaMoS'08)*, 2008
5. Grau G., Franch X., Ávila A.: J-PRiM: A Java Tool for a Process Reengineering i* Methodology. *Proceedings of the 14th IEEE International Requirements Engineering Conference (RE'06)*, 2006
6. Cares C., Franch X., Perini A. and Susi A.: iStarML Reference's Guide. Technical Report. LSI-07-46-R, UPC, Barcelona (2007)