Features as Loosely Defined Method Fragments

Kuldar Taveter
Department of Informatics, Tallinn University of Technology, Tallinn, 12618, Estonia
(+372) 6202 313
kuldar.taveter@ttu.ee

Leon Sterling
Department of Computer Science and Software Engineering, University of Melbourne, Victoria, 3010, Australia
(+61 3) 8344 1404
leonss@unimelb.edu.au

1 Introduction

In [1] is proposed a modular approach enabling developers to build customized project-specific methodologies from agent-oriented software engineering (AOSE) features. An *AOSE feature* is defined in [2] to encapsulate software engineering techniques, models, supporting Computer-Aided Software Engineering (CASE) tools and development knowledge such as design patterns. It is considered a stand-alone unit to perform part of a development phase, such as analysis or prototyping, while achieving a quality attribute such as privacy. Another method for building customized methodologies – OPEN [3] – includes the notions of Work Units, Work Products, Producers, Stages, and Languages. We can define an *AOSE feature* in terms of these notions as a Work Unit performed by one or more Producers in support of a specific software engineering Stage resulting in one or more Work Products represented in the respective Languages. For example, we can identify the feature of goal and role modelling performed by a domain engineer in support of the requirements engineering stage and resulting in goal models and role schemas. Analogously, we can identify the feature of simulation performed by a domain engineer and system architect in support of the rapid prototyping stage and resulting in the executable constructs of the JADE agent platform represented in the Java language. Both of the features mentioned support the quality goals of adequacy and correctness of the requirements captured.

According to [3], a *work product* is any significant thing of value (e.g., document, diagram, model, class, or application) that is developed during a project. A *language* is the medium used to document a work product. A *producer* is anything that produces (i.e., creates, evaluates, iterates, or maintains), either directly or indirectly, versions of one or more work products. A *work unit* is defined as a functionally cohesive operation that is performed by a producer. A *stage* is a formally identified and managed duration of time. Differently from OPEN [3], we do not regard it as necessary to rely on the formal metamodel of method fragments. As has been demonstrated by our earlier work [1, 2, 4], and by the forthcoming book [5], informal approach to methodology composition works equally well and is more likely to be adopted by industry.

2 The conceptual space

In place of a formal metamodel, we define features based on an ontologically-founded conceptual space within which to view systems. Two kinds of entities inhabit the conceptual space: abstract entities and concrete entities. *Abstract entities* are entities that exist neither in space nor in time, that is they cannot be localized. Examples of abstract entities are mathematical objects like numbers and sets, *modelling abstractions* like goals and roles, as well as *types*. On the other hand, *concrete entities* are defined as entities that exist at least in time. They subsume physical entities that exist in both time and space, and virtual entities that exist merely in time. Examples of physical entities are humans and machines and as examples of virtual entities serve actions and events. Likewise, a software system exists in time but can we claim that it also exists in space?

The conceptual space consists of three layers: a *motivation layer*, a *system design layer*, and a *deployment layer*. These layers straightforwardly correspond to the three kinds of models defined by the Model-Driven Architecture (MDA) by the Object Management Group (OMG): Computation-Independent Models (CIM), Platform-Independent Models (PIM), and Platform Specific Models (PSM). The conceptual space is represented in Figure 1. The CIM or motivation layer contains abstract modelling concepts needed for defining requirements and purposes of a system. Arguably, the most foundational are the *goals* of the system, which must be modelled, as well as *roles* for achieving the goals. Here goals represent functionalities expected from the system, while roles are capabilities of the system required for achieving the functionalities. In addition, *social policies* are domain-specific guidelines on interaction and behaviour of agents playing the roles. *Domain entities* are the basic concepts of the problem domain of the system.

The PIM or system design layer consists of the notions required for modelling and designing a socio-technical system. The central one among them is the concept of *agents*, which is depicted at the PIM or system design layer in Figure 1. We define an agent as an autonomous entity situated in an environment capable of both perceiving the environment and acting on it. Each agent belongs to some *agent type* that, in turn, is related to one or more roles from the CIM layer. Agents enact roles by performing *activities*. Each activity instantiates some *activity type* that specifies and refines functionalities defined by goals at the CIM layer. Activities are started and sequenced by *rules*. Rules thus determine *when* goals are to be achieved. Rules are triggered by *perceptions* of events by an agent and/or by the *knowledge* held by an agent. The knowledge consists of a set of *knowledge items* where each knowledge item belongs to a specific *knowledge item type*. An activity consists of *actions* where each action is of some *action type*.

The environment is populated by *concrete agents* and *concrete objects*, which are shown at the PSM or deployment layer in Figure 1. Concrete agents and objects belong to the respective *concrete agent types* and *concrete object types*, such as agent and object types of a specific software platform. They are derived from the agent types and knowledge items of the PIM layer. Likewise, *behavioural construct types* of the PSM layer are based on the rules of the PIM layer. Behavioural construct types are instantiated by the corresponding *behavioural constructs*. At the PSM layer of the conceptual space, the counterparts of action types of the PSM layer are *concrete action types*. A concrete action performed by one agent can be perceived as an *event* by other agents. Events belong to *event types*.

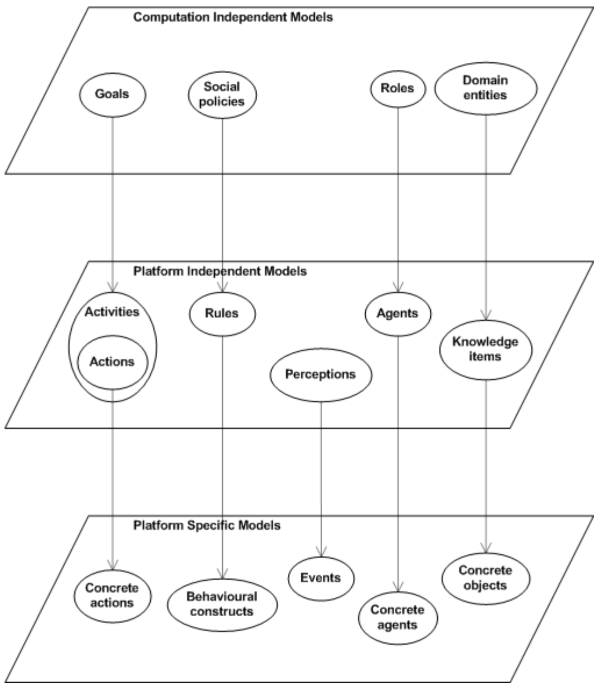

Figure 1. The conceptual space.

Orthogonal to the three horizontal layers are vertical concerns that cross the layers. They are needed for a clear understanding of the issues to be addressed when designing and implementing a system. The types of models required for domain analysis, system design, and implementation lie at the intersections of abstraction layers and cross-cutting concerns. The kinds of such vertical concerns are provided by *conceptual frameworks*. In the forthcoming book [5], we provide an overview of the existing conceptual frameworks and then describe our own conceptual framework – the *viewpoint framework* – suitable for modelling distributed socio-technical systems. This conceptual framework has three vertical modelling aspects: *interaction*, *information*, and *behaviour*. After dividing the conceptual space vertically into the three modelling aspects, we define and explain ten types of models. Each of these model types fits into a particular compartment of the conceptual space. *Goal models* define the system's purpose by the goals and quality goals set for it. Goal models include roles, which define functionalities required for achieving the goals. *Motivational scenarios* describe in an informal and loose narrative manner how goals are to be achieved by agents enacting the corresponding roles. *Role models* describe responsibilities and constraints related to the performing of particular roles and *organisation models* describe relationships between roles. *Domain models* describe the knowledge that the system is supposed to handle. The purpose of *agent models* is to transform the abstract constructs from the analysis stage – roles – to concrete constructs – *agent types* – that will be realized at runtime. The *acquaintance model* complements the agent model by outlining interaction pathways between the agents of the system. *Interaction models* represent interaction patterns and protocols between the agents. *Knowledge models* describe common and private knowledge for the agents. *Service models* describe physical and virtual environments that have been made computationally accessible by agents – computational environments. Finally, *behaviour models* describe the process of decision-making and performing activities by individual agents. *Platform-specific models* related to a specific software platform, such as Microsoft.NET or JADE, can also be utilized.

3 Identification of features

Implicit in the conceptual space introduced in Section 2 are three Stages: Analysis, Design, and Implementation. If needed, they can be split into sub-stages like requirements engineering and architectural design. A model of each generic model type described in Section 2 results from performing a Work Unit by some Producer. Models themselves are Work Products represented in the corresponding Languages. Table 1 represents the Stages, Work Units, Producers, Work Products, and Languages present in the combination of models originating in the ROADMAP (Role-Oriented Analysis and Design for Multiagent Programming) and RAP/AOR (Radical Agent-Oriented Process / Agent-Object-Relationship) AOSE methodologies. These methodologies are described in [8] and [9], respectively. As we described in Section 1, features are orthogonal to the notions of OPEN. Relying on the definition of a feature provided in Section 1, we can identify ten features in Table 1. These features have been utilized in the Intelligent Lifestyle project [6] conducted at the University of Melbourne.

Table 1. Stages, Work Units, Producers, Work Products and Languages of the Intelligent Lifestyle project.

Stage	Work Units	Producer(s)	Work Product(s)	Language(s)
Conceptual domain modelling	Goal and role modelling, organisation modelling, conceptual interaction modelling, domain modelling	Domain engineer	Goal models and motivational scenarios, role schemas, agent diagram, interaction-frame diagram, domain model	ROADMAP Graphical Modelling Language
Platform-independent computational design	Agent modelling, acquaintance modelling, interaction modelling, knowledge modelling, behaviour modelling	Domain engineer, system architect	Interaction-sequence diagrams, agent diagram, scenarios, behaviour diagrams	AOR Modelling Language
Platform-specific design and implementation	Implementing agents	Software engineer	Constructs of JADE [7] agent platform	Java

4 Conclusions

We proposed features as loosely defined method fragments. We do not regard it as necessary to rely on the formal metamodel(s) for method engineering, because we have seen that informal approach to methodology composition works equally well and is more likely to be adopted by industry. In the near future, we plan to come up with process descriptions for method engineering based on the conceptual space.

5 References

[1] Juan, T., Sterling, L., and Winikoff, M. (2002). Assembling agent oriented software engineering methodologies from features. In F. Giunchiglia, J. Odel, and G. Weiss (Eds.), *Agent-Oriented Software Engineering III, Third International Workshop, AOSE 2002, Bologna, Italy, July 15, Revised Papers and Invited Contributions* (LNCS 2585, 198–209). Berlin, Germany: Springer-Verlag.

[2] Juan, T., Sterling, L., Martelli, M., and Mascardi, V. (2003). Customizing AOSE methodologies by reusing AOSE features. In J. S. Rosenschein, T. Sandholm, M. Wooldridge, and M. Yokoo (Eds.), *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-03), Melbourne, Australia, July 14–18* (113–120). New York, NY: ACM.

[3] Firesmith, D. G., and Henderson-Sellers, B. (2002). *The OPEN process framework: An introduction*. London, UK: Addison-Wesley.

[4] Juan, T., and Sterling, L. (2003). The ROADMAP meta-model for intelligent adaptive multiagent systems in open environments. In P. Giorgini, J. Muller, and J. Odell (Eds.), *Agent-Oriented Software Engineering IV, 4th International Workshop, AOSE 2003, Melbourne, Australia, July 15, Revised Papers* (LNCS 2935, 53–68). Berlin, Germany: Springer-Verlag.

[5] Sterling, L. & Taveter, K. (2009). *The art of agent-oriented modeling*. Cambridge, MA; London, England: The MIT Press (forthcoming).

[6] Sterling, L., Taveter, K., and the Daedalus Team (2006). Building agent-based appliances with complementary methodologies. In E. Tyugu and T. Yamaguchi (Eds.), *Knowledge-Based Software Engineering, Proceedings of the Seventh Joint Conference on Knowledge-Based Software Engineering* (JCKBSE ’06), August 28–31, Tallinn, Estonia (223–232). Amsterdam, The Netherlands: IOS Press.

[7] Bellifemine, F., Caire, G., and Greenwood, D. (2005). *Developing multiagent systems with JADE*. Chichester, UK: John Wiley and Sons.

[8] Juan, T., and Sterling, L. (2003). The ROADMAP meta-model for intelligent adaptive multiagent systems in open environments. In P. Giorgini, J. Muller, and J. Odel (Eds.), *Agent-Oriented Software Engineering IV, 4th International Workshop, AOSE 2003, Melbourne, Australia, July 15, Revised Papers* (LNCS 2935, 53–68). Berlin, Germany: Springer-Verlag.

[9] Taveter, K., and Wagner, G. (2005). Towards radical agent-oriented software engineering processes based on AOR modeling. In B. Henderson-Sellers and P. Giorgini (Eds.), *Agent-Oriented Methodologies* (277–316). Hershey, PA: Idea Group.