DFKI Technical Report

A Methodology for Emergent Software

September 2010, Daniel SONNTAG
German Research Center for Artificial Intelligence, Stuhlsatzenhausweg 3, D-66123 Saarbruecken, Germany

Abstract. We present a methodology for software components that suggests adaptations to specific conditions and situations in which the components are used. The emergent software should be able to better function in a specific situation. For this purpose, we survey its background in metacognition and introspection, develop an augmented data mining cycle, and invent an introspective mechanism, a methodology for emergent software. This report is based on emergent software implementations in adaptive information systems (Sonntag, 2010).

Keywords. Emergent Algorithms, Introspection, Embedded Data Mining, Meta Architectures, Adaptivity

1. Introduction

When you search for the term “emergent software” on prominent Internet search engines, you realise that the term is not used as expected. It often refers to neural network software intended for creating complex models of the brain and cognitive processes. The literal meaning of emergent—arising as a natural or logical consequence—however, is excluded to a great extent. The term “emergent algorithm” exists in Wikipedia and denotes an algorithm that (1) achieves predictable global effects, (2) does not require global visibility, (3) does not assume any kind of centralised control, and (4) is self-stabilising.

Our new methodology for emergent software, which is explained in this report, uses the first and last of the aforementioned characteristics to form a new, technically and procedurally driven definition of emergent software. Essentially, this definition relies on the psychologically motivated field of introspection as a theoretical basis and includes embedded data mining as the major technical component. Additionally, the procedure of *emergence* should be understood as the competence to adapt to specific conditions in specific situations (thereby, we are referring to the technical context, not the physical environment). This leads us to the following definition:

An emergent software is an embedded software component that is able to monitor and introspect the specific condition in which it is used, builds an own model of this condition, and suggests adaptations to this condition to better function in a similar situation.

The adaptation step itself can either be supervised by a human expert, or the system self-adapts to a specific condition and situation. This report is structured as follows. Before explaining the new methodology of emergent software according to the above defi-

nition in more detail, we will introduce the background of metacognition and introspection. Subsequently, the methodology of emergent software should be implemented by an introspective mechanism. A full implementation and more detailed description of the introspective mechanism can be found in Sonntag (2010), where we discuss ontologies and adaptivity in dialogue for question answering applications.

2. Background of Metacognition and Introspection

Metacognition is cognition about cognition. Humans use *metacognition* to monitor and control themselves, to choose goals, to assess their progress, and to adopt new strategies for achieving goals. Psychological literature emphasises cognitive self-monitoring and the importance of explicit representations (Yussen, 1985). We base our analysis (also cf. (Nelson and Narens, 1990)) of metacognition on three principles:

1. *The cognitive processes are split into two or more interrelated levels.*
2. *The meta-level (metacognition) contains a dynamic model of the object-level (cognition).*
3. *The two dominant relations between the levels are called* **control** *and* **monitoring**.

The basic conceptual structure for two levels, the object-level and the meta-level, is shown in figure 1. We will use this two level structure to model an introspective mechanism for emergent software. Generalisations to more than two levels have been developed in Sonntag (2008).

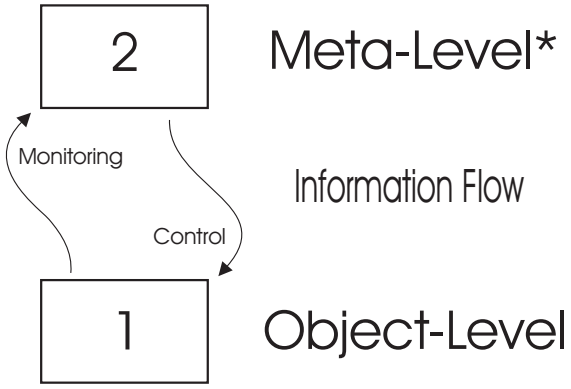

Figure 1. The introspective mechanism for emergent software is based on two conceptual levels, (1) the object-level, and (2) the meta-level, whereby $1 \rightarrow 2$ is an asymmetric relation *monitoring*, and $2 \rightarrow 1$ is an asymmetric relation *control*. Both form the information flow between the two levels; monitoring informs the meta-level and allows the meta-level to be updated. Depending on the meta-level, the object-level is controlled, i.e., to initiate, maintain, adapt, or terminate object-level (cognitive) task-based activities of the emergent software component.

The Metacognition in Computation Symposium (Anderson and Oates, 2005) brought together researchers from many disciplines (computer science, philosophy, psychology) to discuss metacognition in AI systems. Cox (2005) presents an excellently selected research review about metacognition in computation. He introduces cognition

and metacognition in psychological literature which provides a wide array of influences on metacognition in computation, for example how cognitive functions develop during childhood. Tash and Russell (1994) use control strategies for a syntactic planner by applying probabilistic estimations and decision theory to select a strategy with the most expected utility. Derry (1989) associates metacognitive components with the ability of a subject (or intelligent agent in general) to orchestrate and monitor knowledge of the problem solving process. Davidson, Deuser and Sternberg (1994) argue that metacognitive abilities correlate with standard measures of intelligence. Brachman (2002) talks about systems that know what they are doing.

Based on these explanations of metacognition, its relationship to emergent software becomes clear. Metacognition provides many features of our software we would like to include. More precisely, it

1. Provides self-improvement by adaptation and customisation;
2. Offers designs for never-ending learning[1]; and
3. Integrates a variety of previously isolated findings: meta architectures, supervised and unsupervised learning, reinforcement learning, interactive learning, and embedded data mining.

Apart from the complexity, the theory highlights an empirically tractable model creation and verification process. This process has to be embedded into a learning framework for complex software architectures. We used the CRISP Data Mining Cycle and augmented it with the concept of an emergent software component that is able to change its configuration over time.

3. Augmented CRISP Data Mining Cycle

According to control theory, we should not only be able to vary parameters of the object level control in real-time, but augment the object-level (cognitive) reasoning process by learned meta models. Hence, the emergent software idea includes planning, monitoring, authoring, integration/adaptation, and evaluation.

The last two steps, integration/adaptation and evaluation, are implemented by augmenting the data mining life cycle to support a live integration of obtained models. We call this additional step the *(automatic) operationalisation* of learned meta models. Figure 2 illustrates the Cross Industry Standard Process for Data Mining cycle[2] and includes our augmentation. In the *modelling phase*, various modelling techniques are selected and applied. The modelling phase is finished when one or more models, which appear to be of high quality at least from a data analysis perspective, have been built. These models then need to be evaluated before their deployment. In the *evaluation phase* we use the models to review the model building process. This evaluation is done by running the system on unseen supervised data or by performing reinforcement learning experiments. Finally, at the end of the evaluation stage, a decision has to be reached as to whether to use the data mining results obtained. Then a new model is deployed and used in the domain or business units.

[1] *Find a bug in a program, and fix it, and the program will work today. Show the program how to find and fix a bug, and the program will work forever.* (Oliver Sefridge in Hearst and Hirsh (2000))

[2] See *http://www.crisp-dm.org*.

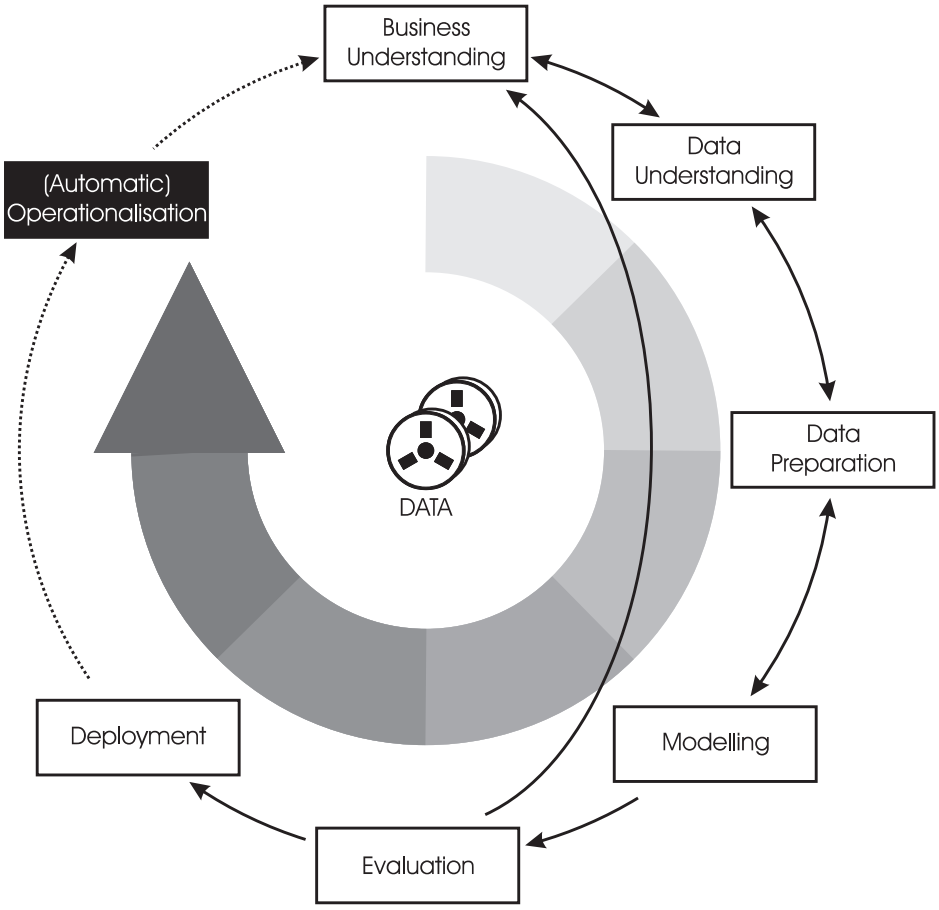

Figure 2. Adapted CRISP data mining cycle. CRISP is characterised by its independence from the application domain and the algorithms used. This makes it suitable as base data mining cycle where emergent aspects are included.

Our aim to integrate the introspective mechanism in order to extend the data mining cycle by a new phase where system introspection is integrated, resulted in a new step of the data mining life cycle, i.e., *(automatic) operationalisation*. The introspective models are directly used in conjunction with the former decision making models for action taking (updating the component's internal reasoning procedure).

It is important to note that empirical machine learning models are pattern matching systems; we expect the behaviour of the system to be improved by drawing an analogy to a past experience which materialises as patterns to be mined. These patterns do not necessarily follow logical rules in terms of a higher order logic—but instead, they should follow at least the causal implications of a propositional logic which helps to implement emergent software aspects based on learned causality. All patterns to be mined can be regarded as *introspective reports* on the application or business domain.

4. Introspective Mechanism—The methodology for emergent software

One of the basic questions of an introspective mechanism in the context of emergent software is how to monitor the system and update the reasoning procedure based on the learned models. Automatic operationalisation allows us to learn adaptation models online and in an incremental way. This corresponds to the definition of emergent algorithms and emergent software—suggesting adaptations in special conditions to better function in similar situations. Likewise, some of the experimental setups may involve direct user feedback as a reinforcement signal in order to select adaptations proposed by the system.

The introspective mechanism is an introspective learning system that uses learning methods in order to build models during the emergent software's performance task.

4.1. Methodology

The methodology consists of

1. metadata management theory;
2. an introspective control model; and
3. a suitable learning environment and learner pool.

By implementing the introspective mechanism, we will emphasise the design of machine learning systems that monitor themselves, self-diagnose, and self-repair in terms of self-adaptation to changing conditions. The introspective mechanism functions as a means to monitor the complete software system to (1) make it easier to discover the errors in processing; and (2) understand the cause of error/failure or success.

4.2. Metadata Management Theory

Metadata can be understood as data about data. The distinction between data and metadata depends on the specific application in mind and the methods used for extracting, annotating, or representing metadata. Many of the complex decisions to be made for emergent algorithmic behaviour can be based on metadata collected during the workflow of processing the software inputs and outputs or the history of last software component invocations.

Metadata are then used as attributes for machine learning experiments. Witten and Frank (2000) point out the existence of basically three kinds of relations between two attributes they named: *semantic* (indicates that two attributes only make sense when used together), *causal* (occurs when one attribute causes another, making a prediction of the latter, only meaningful if the first is included), and *functional* (the functional dependency of one attribute on another means that if the latter is used, e.g., in an association rule, the former becomes superfluous). Instances of these three relations are empirically generated by machine learning algorithms in the learning environment. Their applicability for an emergent behaviour of the software must then be clarified by an introspective control model.

4.3. Introspective Control Model

Introspective control is the application of the introspective knowledge gained from the metadata. We use the introspective mechanism to generate models that can be manually

or (semi-)automatically operationalised. This means that we put an introspective control into effect by exploiting an information state (system monitoring of the emergent software) from which we can extract useful features for model generation (e.g., generating association rules or inducing decision classifiers). The application of the learned models is the control we exert on the emergent software's task level.

1. *Manual Operationalisation*: In manual operationalisation, the system expert is confronted with the machine learning models derived from the exploratory data mining analysis. The system expert thereby inspects the automatically derived models and authors their operationalisation manually, i.e., he adjusts the emergent system according to the expert knowledge he gets by inspecting the data mining results.
2. *Semi-Automatic Operationalisation*: Operationalisation can also be used in an online learning process in which the expert user authors and adjusts the system by using an interactive evaluation environment to sort out, for instance, non-causal from causal rules that the emergent software component generated automatically.
3. *Automatic Operationalisation*: The objective of automatic operationalisation is to automatically apply model changes to prevent failures that became apparent after deployment. Such failures are prevented from recurring when a new CRISP cycle is started and evaluated (instead of "only" providing suggestions to the dialogue experts).

4.4. Learning Environment

Jafar-Shaghaghi (1996) distinguishes between two categories of data mining techniques: statistical and machine learning. While statistical techniques revolves around (non)parametric statistical data analysis, examples for machine learning techniques are neural networks, genetic algorithms, and symbolic learning techniques. Special data mining techniques are decision trees, production rules, and inductive logic programming. When humans interpret the results of data mining (which is normally the case), the induced models are a kind of expert advice. When agents interpret the models automatically, agents become experts themselves.

4.5. Abstract Implementation

As mentioned before, by implementing the methodology/introspective mechanism, we will emphasise the design of machine learning systems that monitor themselves, self-diagnose, and self-repair in terms of self-adaptation to changing conditions. The abstract implementation declares the data structures of the control model and the algorithmic interface which can then be implemented by concrete search and prediction algorithms in the learning environment.

4.6. Control Model

With the help of ontological assertions (Fensel, Hendler, Lieberman and Wahlster, 2003; McGuinness, 2004; Hitzler, Krötzsch and Rudolph, 2009) (i.e., in RDF[3], OWL[4], or

[3] http://www.w3.org/RDF/

[4] http://www.w3.org/2004/OWL/

OWL-2[5]) about the specific domain, e.g., implicit ordering in attribute values, machine learning schemes could use process metadata as essential information for model creation; the solution we propose is to use metadata attributes of propositional logic, with special relationships among them.

Using ontological metadata repositories, the semantic relationships between attributes obtain a clear formalisation. In addition, we obtain a good representational basis for causal attribute relationships. One of the main tasks of effective metadata management is to provide the relevant input spaces derived from ontology instances to build causal attributes to be introduced in the "information state" of the emergent software. These attributes serve as rule items of classification rules in the causal direction. In addition, learning models often reveal causal and functional dependencies the system expert was already aware of. This implies using input spaces which rarely contain functional dependencies so that interesting patterns are not obscured. By carefully labelling propositional information state features, the extracted data only contains features obtained from ontological representations with easily-understandable propositions for directing emergent behaviour.

4.7. Learning Environment

All algorithms should be based on the assumption that any hypothesis found to approximate the target function well over a sufficiently large training set will also do the same over a set of unobserved instances. This assumption is also known as *the inductive learning hypothesis* (Mitchell, 1997), which, roughly speaking, ensures that the machine learning models we create can be used for future decisions in the same subject domain. This means that the emergent behaviour of the software component has a theoretically grounded optimisation criterion: the adaptation according to empirically found learning models of the environment.

We suggest algorithms such as Naive Bayes, decision trees, K-nearest neighbour, and support vector machines for supervised classification; association rules are for unsupervised association pattern analysis. All these algorithms are based on the statistical *maximum likelihood principle*, which corresponds to the decision of a rational emergent software agent, where the decision is based upon (i.e., a class is assigned to) the class with the the greatest probability of occurrence. Each of the possible classes $c_i \in \mathrm{C}$ is considered and the corresponding class-conditional densities $p(x|c_i)$ are estimated for the finite attribute vector x of a training instance. The adaptation of the emergent software component to specific conditions corresponds to the class-conditional densities after the learning step.

4.8. Workflow

Figure 3 outlines the main ingredients of the introspective mechanism / emergent algorithm and the workflow that we aim to achieve, i.e., the formulation of practical introspective methods and their verification by practical machine learning and embedded data mining experiments toward the implementation of manual and (semi-) automatic operationalisation of introspective machine learning models in the context of emergent

[5] http://www.w3.org/TR/owl2-profiles/

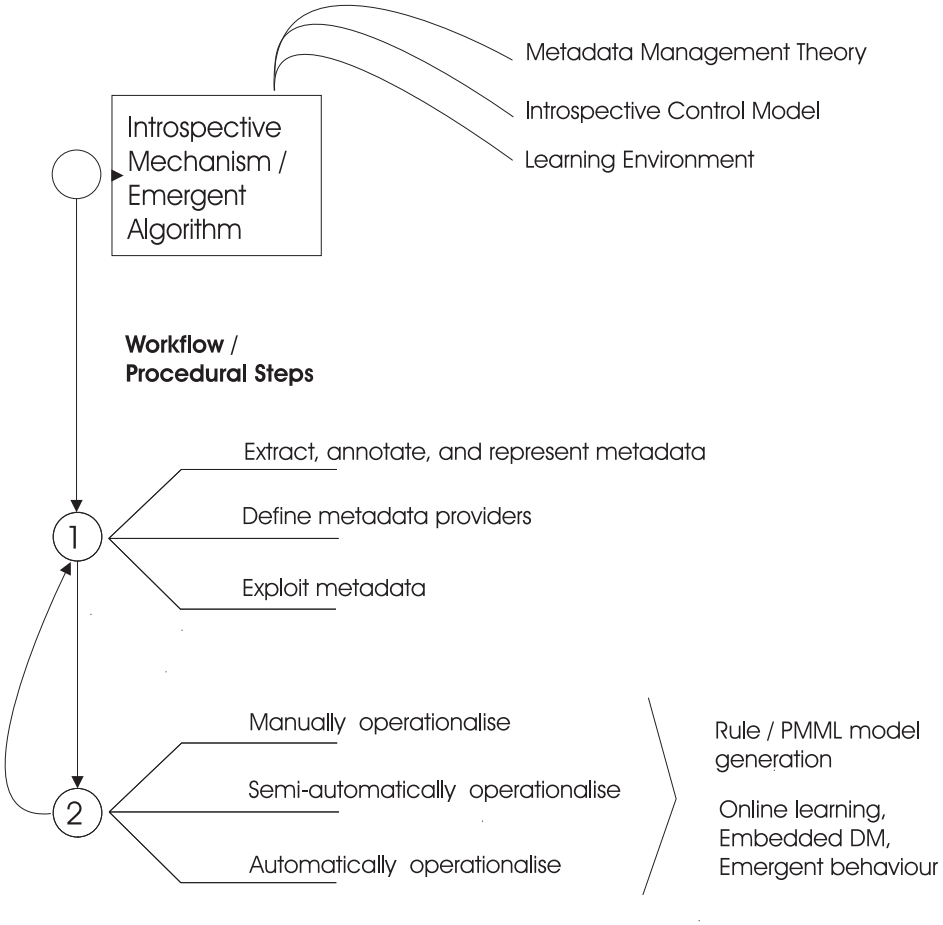

Figure 3. Workflow of implementing steps for emergent software

software systems. The workflow has two steps and finishes with a rule / PMML model generation step.

PMML allows users to develop models within one vendor's application, and use other vendors' applications to analyse and apply the models. With PMML, the exchange of models between compliant applications is possible. This provides a straightforward way to use machine learning models in an embedded environment such as emergent software applications. One or more mining models can be contained in a PMML document. We propose the usage of association rules PMML models, where a set of items is associated with another set of items. The model consists of model attributes, items, item sets, and the association rules. Lavrac (2006) describes PMML as the standard for integrating data mining and decision support technology with information systems.

The rules can then be used to adapt the emergent software component's behaviour by applying them, i.e., updating the component's internal reasoning procedure based on the learned models.

5. Conclusion

We presented a new methodology for developing emergent software. Based on one possible technically and procedurally driven definition of emergent software, we provided the design of an introspective mechanism which includes a metadata management theory, an introspective control model, and a learning environment. Additionally, we provided an abstract implementation of the workflow. A full implementation for a specific application, i.e., self-adaptive behaviour of a dialogue-based question answering system, can be found in (Sonntag, 2010).

Future work should investigate the suitability of different machine learning model languages such as PMML for the reasoning process of embedded emergent software components. The automatic update of the reasoning process remains a critical problem in terms of reliability, trust, and security issues. Additionally, other definitions of emergent software might focus on the mutation aspect of emergence where, e.g., genetic algorithms without pre-specified optimisation criteria should be employed.

Acknowledgments. Daniel Sonntag is supported by the THESEUS Programme funded by the German Federal Ministry of Economics and Technology (01MQ07016).

References

Anderson, M. and Oates, T. (Eds.) (2005). *2005 AAAI Spring Symposium on Metacognition in Computation, March 21-23 2005, Stanford, California, USA*, Volume SS-05-04 of *AAAI Technical Report*. AAAI Press.

Brachman, R. J. (2002). Systems That Know What They're Doing. *IEEE Intelligent Systems 17*(6), 67–71.

Cox, M. T. (2005, December). Metacognition in computation: A selected research review. *Artificial Intelligence 169*(2), 104–141.

Davidson, J., Deuser, R., and Sternberg, R. (1994). *Metacognition*, Chapter The role of metacognition in problem solving., pp. 207–226. The MIT Press.

Derry, S. (1989). *Cognitive strategy research: From basic research to educational applications*, Chapter Strategy and expertise in solving word problems, pp. 269–302. Springer-Verlag, NY.

Fensel, D., Hendler, J. A., Lieberman, H., and Wahlster, W. (Eds.) (2003). *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential*. MIT Press.

Hearst, M. A. and Hirsh, H. (2000). AI's Greatest Trends and Controversies. *IEEE Intelligent Systems 15*(1), 8–17.

Hitzler, P., Krötzsch, M., and Rudolph, S. (2009, August). *Foundations of Semantic Web Technologies*. Chapman & Hall/CRC.

Jafar-Shaghaghi, F. (1996). *Maschinelles Lernen, Neuronal Netze und Statistische Lernverfahren zur Klassifikation und Prognose*. Shaker, Aachen, Germany.

Lavrac, N. (2006). *From Data and Information Analysis to Knowledge Engineering*, Chapter SolEuNet: Selected Data Mining Techniques and Applications, pp. 32–39. Springer.

McGuinness, D. L. (2004, January). Question answering on the semantic web. *IEEE Intelligent Systems 19*(1), 82–85.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill International Edit.

Nelson, T. O. and Narens, L. (1990). *G. H. Bower (Ed.) The Psychology of Learning and Motivation: Advances in Research and Theory*, Volume 26, Chapter Metamemory: A theoretical framework and new findings, pp. 125–169. Academic Press.

Sonntag, D. (2008). On introspection, metacognitive control and augmented data mining live cycles. *CoRR* **abs/0807.4417**.

Sonntag, D. (2010). *Ontologies and Adaptivity in Dialogue for Question Answering*. AKA and IOS Press, Heidelberg.

Tash, J. and Russell, S. (1994). Control strategies for a stochastic planner. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, Volume 2, Seattle, Washington, USA, pp. 1079–1085. AAAI Press/MIT Press.

Witten, I. H. and Frank, E. (2000). *Data Mining*. Morgan Kaufman, San Mateo, CA.

Yussen, S. R. (1985). *The Growth of Reflection in Children*. Academic Press Inc.,U.S.