Producing Software by Integration: Challenges and Research Directions (Keynote)

Paola Inverardi, Marco Autili, Davide Di Ruscio, Patrizio Pelliccione, Massimo Tivoli
Information Engineering, Computer Science and Mathematics department
University of L'Aquila, L'Aquila, Italy
{paola.inverardi, marco.autili,davide.diruscio,
patrizio.pelliccione,massimo.tivoli}@univaq.it

ABSTRACT

Software is increasingly produced according to a certain goal and by integrating existing software produced by third-parties, typically black-box, and often provided without a machine readable documentation. This implies that development processes of the next future have to explicitly deal with an *inherent incompleteness of information* about existing software, notably on its behaviour. Therefore, on one side a software producer will less and less know the precise behaviour of a third party software service, on the other side she will need to use it to build her own application.

In this paper we present an innovative development process to automatically produce dependable software systems by integrating existing services under uncertainty and according to the specified goal. Moreover, we (*i*) discuss important challenges that must be faced while producing the kind of systems we are targeting, (*ii*) give an overview of the state of art related to the identified challenges, and finally (*iii*) provide research directions to address these challenges.

Categories and Subject Descriptors

D.2.0 [**Software Engineering**]: General; K.6.3 [**Software Management**]: Software development; D.2.9 [**Management**]: Software process models

General Terms

Design, Theory

Keywords

Dependable software systems, automated synthesis, model elicitation, model-driven engineering.

1. INTRODUCTION

In the next future we will be increasingly surrounded by a virtually infinite number of software applications that provide computational software resources in the digital space. According to John Musser, founder of ProgrammableWeb[1], the production of application programming interfaces (APIs) growths exponentially and some companies are accounting for billions of dollars in revenue per year via API links to their services. Moreover, the evolution of today Internet is expected to lead to an ultra large number of available services, hence increasing their number from 10^4 services on 2007 to billions of services [62] in the near future.

This situation radically changes the way software will be produced and used:

- first, software is increasingly produced according to a certain goal and by integrating existing software;
- second, it shifts the focus of software production on reuse of third-parties software, typically black-box, that is often provided without a machine readable documentation.

The first characteristic implies a goal oriented, opportunistic use of the software being integrated, i.e. the producer will only use a subset of the available functionalities, some of which may not even be (completely) known. The second one implies the need to (i) extract suitable observational models from discoverable and accessible pieces of software, that are made available as services in the digital space, and (ii) devise appropriate integration means (e.g., architectures, connectors, integration patterns) that ease the collaboration and integration of existing services in a dependable[2] way.

In this paper we refer to an innovative development process, called EAGLE, to automatically produce dependable software systems by integrating existing services under uncertainty and according to the specified goal. EAGLE leverages the model-based software production paradigm and permits to move a step forward to face the *inherent incompleteness of information* about existing software. Moreover, EAGLE adopts an experimental approach, as opposed to a creationistic one, to the production of dependable software. In fact, software development has been so far biased towards a creationist view: a producer is the owner of the artefact, and with the right tools she can supply any needed piece of information (interfaces, behaviours, contracts, etc.). However, the knowledge of a software artefact is limited to what

ESEC/FSE'13, August 18–26, 2013, Saint Petersburg, Russia

http://dx.doi.org/10.1145/2491411.2505428

[1] `http://www.programmableweb.com/`

[2] We refer to the general notion of dependability, as defined by IFIP Working Group 10.4: "*the trustworthiness of a computing system which allows reliance to be justifiably placed on the service it delivers*".

can be observed of it. The more the observations will be powerful and costly the more the knowledge will be deep, but always with a certain degree of uncertainty. Indeed, there is a theoretical barrier that limits, in general, the power and the extent of observations.

The considerations above lead us to identify important challenges that must be faced while producing the kind of systems we are targeting. In this paper we (*i*) discuss these challenges, (*ii*) give an overview of the state of art related to the identified challenges, and (*iii*) propose research directions for addressing them.

Paper structure: Section 2 presents the EAGLE development process, by focusing on its phases and challenges that need to be addressed to realize the EAGLE development process. Section 3 presents an explanatory example to make the EAGLE vision concrete. Sections 4, 5, and 6 present the state of the art and research directions for the identified challenges. Finally, Section 7 concludes the paper.

2. EAGLE DEVELOPMENT PROCESS

The big challenge the EAGLE process (first results might be found in [2]) aims at addressing is to live up with the evidence that this immense software resources availability corresponds to a lack of knowledge on the software, notably on its behaviour. A software producer will less and less know the precise behaviour of a third party software service, nevertheless she will use it to build her own application. This very same problem recognized in the software engineering domain [32] is faced in many other computer science domain, e.g., exploratory search [69] and search computing [16].

2.1 EAGLE Phases

Once requirements or user needs, i.e., the goal G, have been specified, EAGLE *explores* available software and makes explicit the degree of uncertainty associated with it in relation to G, and *assists* the producer in creating the appropriate integration means towards G. The goal G can be specified in different ways depending, e.g., on the technical requirements on the software-to-be and assumptions on its environment. In any case, for the goal validation and integration phases to be automated, a goal G specification for EAGLE is a machine-readable model achieved by the producer through an operationalization of the needs and preferences of the user [66]. In the following we focus on the two phases that compose EAGLE: **elicit** and **integrate**.

- **Elicit**: given a software service S, elicitation techniques must be defined to produce models as much complete as possible with respect to an opportunistic goal G. This means that we admit models that may exhibit a high degree of incompleteness, provided that they are accurate enough to satisfy user needs and preferences.

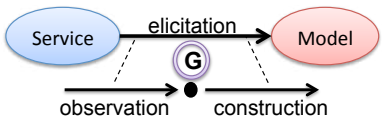

Figure 1: Elicit phase

As shown in Figure 1, the elicit phase is composed of two steps, namely observation and construction [3]. Observation, driven by G, produces a set of observations of the system. In EAGLE we focus on observations corresponding to identify a set of functional or non functional behaviours of the system under analysis, e.g., system response time once executed with a provided input in the execution environment. Construction, driven by G, takes as input the set of observations and produces a system model. This model contains the observed behaviours and typically enriches them with an inference step. Ideally, the elicit phase produces a set of models that once integrated are correct and complete with respect to the goal G, i.e., G holds on the model obtained by integrating the elicited models iff it holds on S, see Figure 2.a. Unfortunately, correctness and completeness of the models cannot in general be achieved. The real situation is shown in Figure 2.b in which models are neither correct nor complete. This is because the set of observations is always finite and typically the elicitation phase has to make an inference step to produce the model.

- **Integrate**: the integrate phase assists the producer in creating the appropriate integration means to compose the observed software together in order to produce an application that satisfies G.

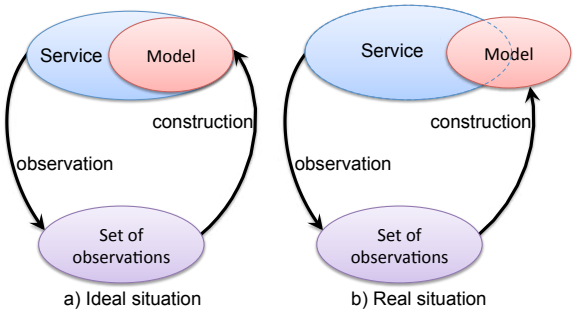

Figure 2: Service, observations, and model

Referring to Figure 3, M_1, M_2,..., M_n obtained through the elicit phase represent models of the services to be integrated; each of these models exhibits its own degree of uncertainty μ_{M_1}, μ_{M_2},..., μ_{M_n}, respectively. For each service multiple models may exist (e.g., behavioural, stochastic or Bayesian), each representing a specific view of the services. Model transformation techniques will ensure coherence and consistency among the different views of the system, hence providing an adequate and systematic support to *model interoperability* [35]. These models are the input of model synthesis techniques together with the goal G. Suitably instantiating architectural patterns and styles [60] and integration patterns [36], the output is an Integration Architecture (IA) that interrelates the elicited models together with additional integrator models as synthesized by EAGLE. Integrator models, besides guaranteeing correctness of the interaction logic, e.g., deadlock freeness and performance system requirements, can compensate the lack of knowledge of the composed

software by also adding extra logic like connectors, mediators and adapters [61, 37, 46], hence enhancing dependability. IA plays a crucial role in influencing the overall uncertainty degree of the final integrated system S, as different IAs may result in different uncertainty degrees for S, namely uS. Once obtained an integration architecture, *code synthesis techniques* provided by EAGLE generate integration code that guarantees, during the system lifetime, the specified goal under a controlled uncertainty degree.

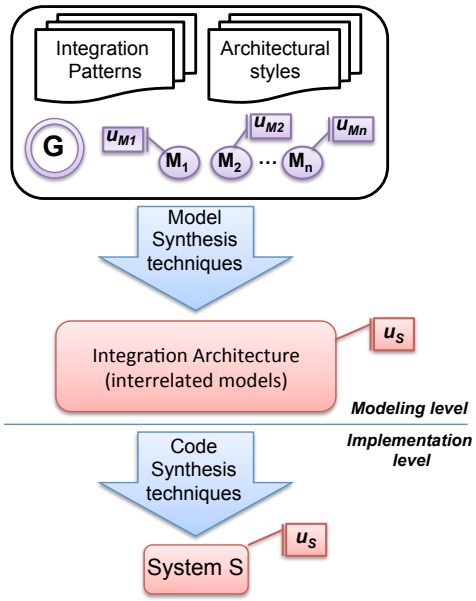

Figure 3: Integration activities

Section 2.2 describes the EAGLE challenges that emerge from the discussion above. Specifically, the three challenges are: *elicitation techniques*, *integrator synthesis*, and *model interoperability and code generation*. Each of these challenges is detailed in Sections 4, 5, and 6.

2.2 EAGLE Challenges

The EAGLE development process poses three challenges that need to be further investigated. Those challenges are:

- **Elicitation techniques**: EAGLE asks for elicitation techniques to automatically explore the available software, extract observations, and produce models that, according to a given goal, abstract the actual runtime behaviour with the best possible accuracy. We target observations on black-box software and we are primarily interested in extracting behavioural models of the software interaction protocols. Moreover, we are interested in providing a measure of the goodness of the model with respect to the system and to the goal that the system has to achieve. Indeed, the elicited models can be incomplete and/or inaccurate where incompleteness refers to the behavioural modelling, i.e., less and/or more traces; and the inaccuracy refers to the quantitative modelling, i.e., inaccurate probabilities and/or quantitative indices [48].

- **Integrator synthesis**: EAGLE requires synthesis techniques to support the automated construction, from the elicited models, of integration means that are correct with respect to guaranteeing the specified goal under a controlled uncertainty degree. In order to be able to make the scenario described above real, several challenges, with respect to the state of the art, need to be addressed. One strand of research concerns synthesis techniques to address the problem of inferring integration extra-logic that aims at solving unexpected issues due to the incompleteness of the models the synthesis process reasons on.

- **Model interoperability and code generation**: This objective aims at generating the skeleton code of the integrator starting from the models produced by the integrator synthesis activity. Since the generation of integration code takes into account all the models of the services composing the system, there is the need for techniques supporting the specification of model interdependencies, and their retrieval. The management of model interdependencies may account also for interoperability aspects between the different notations used for specifying the service models [29]. The generation of code for software artefacts, which are "correct-by-construction", and are able to integrate third-parties software according to their elicited and interrelated models requires dedicated techniques to identify the parts of the generated code, which cannot be modified by developers to not invalidate the goal and the architectural choices. Advanced techniques are also required to support the implementation of correct custom code, which has to be added to complete the generated skeleton.

3. EXPLANATORY EXAMPLE

To make this vision more concrete let us discuss an hypothetical scenario of EAGLE at work. Let us assume that a software producer aims at realizing a mobile application for transportation, App, whose interaction behavior, modeled as a probabilistic automaton and provided by the developer, is shown in Figure 4.e. This automaton, together with the property of interest (Figure 4.f) detailed later, represent the goal to be achieved. The services that will interact with App are:

1. TC_WS, a web service providing the timetable of a Transportation Company (TC) and also a service for subscribed clients to notify their presence on a given bus by means of a check-in/check-out system.

2. Bus, a system composed of two Near Field Communication (NFC) endpoints (one for checking in, one for checking out), a GPS receiver, and a component for notifying to TC time and position of a client;

3. GPS, the well-known service that allows a device to know its geographic position;

4. PaypalWS, the well-known web service for seamless payment processing.

The main feature of the to-be App application is that it will allow clients to purchase virtual tickets by checking-in

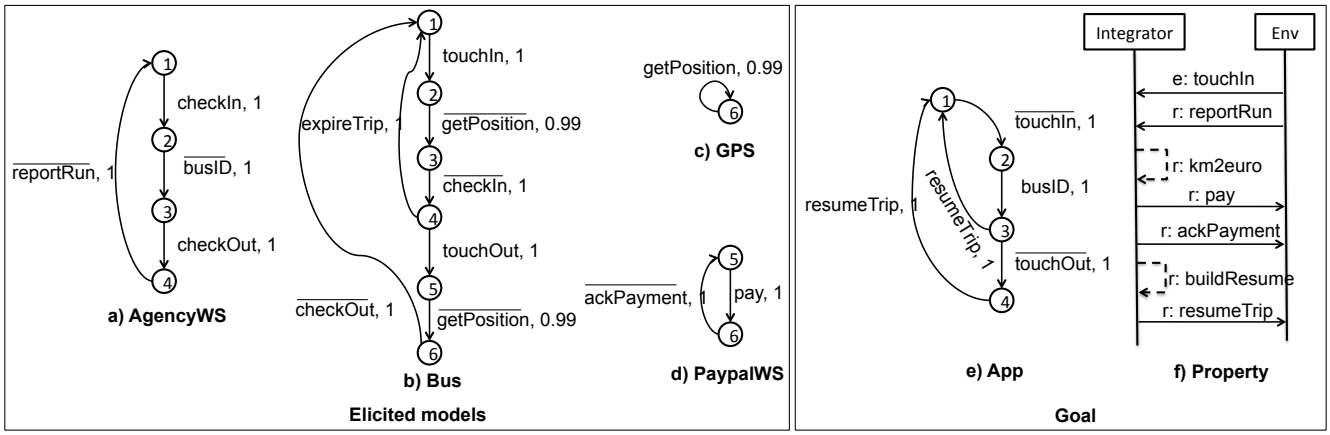

Figure 4: Example - elicited models and goal

on the bus; when the client gets off the bus a check-out operation via NFC is sent and the system will trigger the automatic payment of the virtual ticket via Paypal. As a possible technique to be used in the elicit phase, we consider a to-be StrawBerry tool [9] that, for the EAGLE purposes, is enhanced in order to deal with the uncertainty degree of the elicited models. The current version of StrawBerry produces a set of finite state automata modeling the interaction protocols that must be followed in order to have a correct interaction among the observed services. For the sake of the scenario, the enhanced version of StrawBerry shall produce the probabilistic automata shown in Figure 4.(a-d). Bus can interact with an NFC receiver by receiving a TouchIn message; then, the geographical position of the client is obtained by the GPS service and a checkIn message is sent to TC_WS. The special case in which a client checks in and then forgets to check out is handled by a dedicated operation, triggered by the expireTrip message, which resets the state of Bus; in this case the client pays a default amount of money. The message getPosition has a probability 0.99 to be sent and enables state 3 (e.g., the case of successful geographical localization). The incompleteness of the model concerns the remaining cases in which getPosition is sent with a probability of 0.01. In these cases, the model does not express what the behavior of Bus may be, i.e., which states may be reached (e.g., when trying to send a checkIn message without geographical information). The reason for this incompleteness of the model may be inherent to the service or depend on limits of the elicitation process.

TC_WS upon reception of the CheckIn message, containing the timestamp and geographical location of the client, sends back a unique identifier associated to the bus (this is done for information purposes only). When the client is getting off the bus, a checkOut message is sent to the TC_WS, which in turn will send reportRun message containing all the information regarding the specific run. The observed PaypalWS web service is straightforward: it receives payment requests and then it responds with a notification of the occurred payment via a message called ackPayment.

The integrate phase synthesizes the Integrator for the system composed of TC_WS, Bus, GPS, PaypalWS and App. Furthermore, concerning the interaction between the Integrator to be synthesized and its environment (i.e., all the

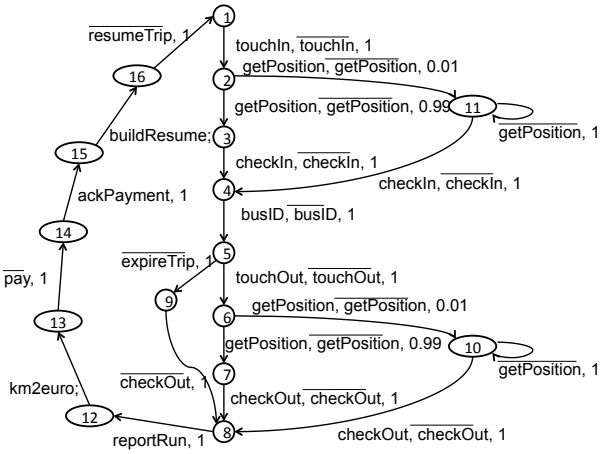

Figure 5: Synthesized integrator model

other components in the system), the integrated system has to satisfy the behavioral property shown in Figure 4.(f) and expressed by using the PSC notation [5]. Informally, this property represents a business requirement for the system stating that whenever a client checks-in on the bus then she has to eventually pay for a virtual ticket and receive proper notifications. Moreover, by referring to the messages km2euro and buildResume shown in Figure 4.(f) the property also expresses requirements on extra logic to be performed by the Integrator. In particular, km2euro represents the logic required to establish the price of the virtual ticket out of the distance covered by the bus. buildResume represents the logic needed to combine the information in reportRun and ackPayment in order to produce resumeTrip. Regarding these messages the Integrator's code to be synthesized can only be a skeleton code to be completed by the producer. Figure 5 shows the probabilistic automaton synthesized for the Integrator (labels are <input>,<output>,<probability>). It suitably mediates the interactions between App and the considered services in order to achieve the specified business requirement while also solving the uncertainty degree introduced by the incomplete-

ness of the elicited Bus model. When the interaction with GPS fails the Integrator applies a Retry strategy to eventually perform a successful geographical localization.

The following three sections report on the state of the art concerning the main areas of interest for EAGLE and, for each area, discuss research directions that can be undertaken to address the identified challenges.

4. ELICITATION TECHNIQUES

4.1 State of the Art

Within the domain of Software Engineering, the early work in 1972 by Biermann and Feldman [10] inspired numerous efforts to infer Finite State Machines (FSMs) of software systems, such as [22, 1, 25]. Several approaches have recently addressed the problem of deriving partial behavioural models from implemented systems. Few approaches have tackled the problem in the setting of black box components, such as [9, 54]. In [9], we propose a method, called StrawBerry, to automatically derive the behaviour protocol of a web-service out of its WSDL interface. The method combines synthesis and testing techniques: the automaton is synthesized based on syntactic data type analysis, and its conformance to the implementation of the corresponding web-service is checked and refined by means of testing. In [54] the authors propose LearnLib, which is a framework for automata learning and experimentation. Active automata learning tries to automatically construct a finite automaton that matches the behaviour of a given target automaton on the basis of active interrogation of target systems and observation of the produced behaviour.

The authors of [47] describe a learning-based black-box testing approach in which the problem of testing functional correctness is reduced to a constraint solving problem. A general method to solve this problem is presented and it is based on function approximation. Functional correctness is modeled by pre- and post-conditions that are first-order predicate formulas. A successful black-box test is an execution of the program on a set of input values satisfying the pre-condition, which terminates by retrieving a set of output values violating the post-condition. Black-box functional testing is the search for successful tests with respect to the program pre- and post-conditions. As coverage criterion the authors formulate a convergence criterion on function approximation. Their testing process is an iterative process. At a generic testing step, if a successful test has to be still found, the approach described in [47] exploits the input and output assignments obtained by the previous test cases in order to build an approximation of the system under testing and try to infer a valid input assignments that can lead the system to produce an output either violating the post-condition or useful to further refine the system approximated model.

The work described in [33] (i.e., the SPY approach) aims to infer a formal specification of stateful black-box components that behave as data abstractions (Java classes that behave as data containers) by observing their run-time behavior. SPY proceeds in two main stages: first, SPY infers a partial model of the considered Java class; second, through graph transformation, this partial model is generalized to deal with data values beyond the ones specified by the given instance pools. The inferred model is partial since it models the intentional behavior of the class with respect to only a set of instance pools provided as input, which are used to get values for method parameters, and an upper bound on the number of states of the model. The model generalization is based on two assumptions: (i) the value of method parameters does not impact the implementation logic of the methods of a class; (ii) the behavior observed during the partial model inference process enjoys the so called "continuity property" (i.e., a class instance has a kind of "uniform" behavior). In our context, we cannot rely on the previously mentioned assumptions.

In [63], the authors propose a novel synthesis technique that constructs partial behavioural models in the form of Model Transition Systems from a combination of safety properties and scenarios. In [43], the authors describe a technique, called GK-Tail, to automatically generate behavioral models from (object-oriented) system execution traces. GK-Tail assumes that execution traces are obtained by monitoring the system through message logging frameworks. For each system method, an Extended Finite State Machine (EFSM) is generated. It models the interaction between the components forming the system in terms of sequences of method invocations and data constraints on these invocations. The correctness of these data constraints depends on the completeness of the set of monitored traces with respect to all the possible system executions that might be infinite. Furthermore, since the set of monitored traces represents only positive samples of the system execution, this approach cannot guarantee the complete correctness of the inferred data constraints.

The approach described in [68], called Jadet, analyzes Java code to infer sequences of method calls. These sequences are then used to produce object usage patterns that serve to detect object usage violations in the code. The approach is white-box and focuses on modeling objects from the point of view of single methods. The work in [24] presents TAUTOKO, which is an approach that, through a combination of systematic test case generation and typestate mining, infers models of program behaviour. The generation of test cases permits to cover previously unobserved behaviour, and systematically extends the execution space, and enriches the specification. The authors of [8] present an approach for inferring state machines with an infinite state space. More precisely, by observing the output that the system produces when stimulated with selected inputs, this work extends existing algorithms for regular inference, which infer finite state machines, to deal with infinite-state systems.

Indeed, the main problem with all these approaches is to assess their goodness. A useful basis for empirically comparing candidate techniques has been provide in a competition to spur the development of inference techniques for FSMs of software systems [67]. The work in [42] presents an empirical comparative study between techniques that infer simple automata and techniques that infer automata extended with information about data-flow. In EAGLE the problem of providing methods and metric to express the accuracy of a model with respect to the system and the goal is of primary importance. An attempt in this direction can be considered [34] where for a white box component it is generated a three-valued interface LTS that explicitly labels states as unknown to reflect the fact that the given sequence of methods invocations leads to a component state that the analysis could not mark as safe or unsafe.

As far as the use of partial behavioural models is concerned, it should be noted that the degree of uncertainty in behavioral models may heavily affect the capability of non-functional analysis techniques. Indeed non-functional (e.g. performance, reliability) models take most of their structure and parameters from software behavior representation. However, this problem is not new, and it has been mitigated by the wide experience in using (in this domain) stochastic models suited for representing uncertainty. On top of these models, several techniques have been recently proposed to validate non-functional attributes of software under uncertainty [48]. The assume-guarantee approaches, typically adopted in the functional world, translate to bayesian approaches in the non-functional world. In fact, bayesian probabilities enable stochastic models to be "conditioned" to specific events that, in turn, have their own probability distributions. Hence, bayesian models (such as Bayesian Networks [50]) can be considered as the stochastic counterpart of the assume-guarantee paradigm. In this direction, an example of bayesian approach for modeling the reliability of a software component-based system, given the reliability of its components, has been presented in [57]. More sophisticated stochastic models can be used to take into account uncertainty in non-functional validation processes. Hidden Markov Models (HMM) [31] are typically used to model systems that have markovian characteristics in their behavior, but they also have some states (and transitions) for which only limited knowledge is available. An example of approach based on HMM that aims at evaluating the reliability of a software component with partial knowledge of its internal behavior has been provided in [17].

Finally, only few approaches initiated the treatment of data, and their synthesis although there is no clear evidence on the effectiveness of their use regarding the goodness of the generated model [42]. The empirical results of this work shows the tradeoffs between the considered techniques. The discussion that is provided in the paper gives indications that can help software engineers in the choice of proper model inference solutions.

A more complete discussion about the state of the art in elicitation techniques might be found in [4].

4.2 Research Directions

As highlighted in the state of the art, the main problem of existing techniques concerns the lack of a precise estimation of their goodness. Moreover, no work in the literature addresses the problem of automatically eliciting quantitative models as well as the possibility to elicit partial models referring to a specific goal. An interesting work in this direction is [9]; it makes use of (interface) data flow analysis, synthesis, and testing to elicit the interaction protocol of a web service. Data flow analysis here plays a key role because it drives the tests selection. We envisage that this same strategy, i.e., coupling static analysis at the interface level with test selection, can be used also to obtain partial models relevant to the goal as well as quantitative models. Goodness of the model can then be defined in terms of the success rates of the three steps, analysis, synthesis and testing. It is worth recalling that in the scope of EAGLE the problem of assessing goodness of a model is integration context-sensitive thus making the problem more tractable.

Notions and metrics to specify the uncertainty degree of models have the following requirements:

1. Introducing different notions of coverage, pivotal to metrics, for defining and quantifying the uncertainty degree of models. Those notions of coverage allow for assessing the effectiveness of the different elicitation techniques and hence of the incompleteness and inaccuracy of the elicited models. For instance, referring to the example in Figure 4, the integrator model shown in Figure 5, which is built in the integration phase, aims at coping with the inherent uncertainty of the elicited service models. This is done by adding to the model suitable arcs, together with their probability, whose aim is to decrease the model incompleteness. The set of behaviours enabled by these additional arcs together with their quantitative properties can be considered as a measure of the model's uncertainty degree.

2. Identifying the portion of the goal specification that can be fulfilled by the system under exploration. It is of paramount importance to identify the role of the pieces of software under exploration during the integration phase and in relation with the goal specification.

3. Providing value based mechanisms to select the exploration techniques and the strategy for their usage according to costs and uncertainty degrees of elicited models.

5. INTEGRATOR SYNTHESIS

5.1 State of the Art

Interoperability and mediation have been investigated in several contexts, among which integration of heterogeneous data sources [70], architectural patterns [56], patterns of connectors [58], Web services [41, 39], and algebra to solve mismatches [30]. Here we discuss the works, from the different contexts, closest to the synthesis methods and techniques that we develop within the EAGLE integrate phase. The interoperability/mediation of protocols have received attention since the early days of networking. Indeed many efforts have been done in several directions including for example formal approaches to protocol conversion, like in [14, 40].

The seminal work in [71] is strictly related to the notions of integrator introduced by EAGLE. Compared to our integrator synthesis approach, this work does not allow to deal with the automated inference of the required extra-logic or with the heterogeneity of the considered services, e.g., heterogeneous interfaces and, hence, different granularity of the elicited models' languages. Recently, with the emergence of web services and advocated universal interoperability, the research community has been studying solutions to the automatic mediation of business processes [65, 64]. However, most solutions are discussed informally, making it difficult to assess their respective advantages and drawbacks.

In [58] the authors present an approach for formally specifying connector wrappers as protocol transformations, modularizing them, and reasoning about their properties, with the aim to resolve component mismatches by also considering the introduction of suitable extra logic.

In [11] the authors present an algebra for five basic stateless connectors that are symmetry, synchronization, mutual exclusion, hiding and inaction. They also give the operational, observational and denotational semantics and a complete normal-form axiomatization. The presented connectors can be composed in series and in parallel. Although

these formalizations supports connector modularization and, hence, compositionality of the synthesis process, automated synthesis is not treated at all hence keeping the focus only on the design and specification of the integration means.

In [52], the authors use a game theoretic approach for checking whether incompatible component interfaces can be made compatible by inserting a converter between them which satisfies specified requirements. This approach is able to automatically synthesize the converter. In contrast to the EAGLE integrator synthesis, this method needs as input a deadlock-free specification of the requirements that should be satisfied by the adaptor, by delegating to the software producer the non-trivial task of specifying that.

In other work in the area of component adaptation [15], it is shown how to automatically generate a concrete adaptor from:

- a specification of component interfaces,
- a partial specification of the components interaction behavior,
- a specification of the adaptation in terms of a set of correspondences between actions of different components and
- a partial specification of the adaptor.

The key result is the setting of a formal foundation for the adaptation of heterogeneous components that may present mismatching interaction behavior. Assuming a specification of the adaptation in terms of a set of correspondences between methods (and their parameters) of two components requires to know many implementation details (about the adaptation) that we do not want to consider in order to synthesize an integrator.

Within the SYNTHESIS [61, 6] and CONNECT [37, 38] projects, we show how to automatically derive either a centralized or distributed connector from a specification of the components' interaction and of the requirements that the composed system must fulfil. However, these approaches do not take into account uncertainty and non-functional requirements of the system to be integrated.

Within the CHOReOS project [7], we are proposing a distributed synthesis approach to automatically derive software entities, called coordination delegates, which implement the logic to integrate and coordinate a set of software services according to a BPMN2 choreography specification. The latter is a model of the goal derived by operationalizing domain expert and user requirements, after being transformed into CTT (ConcurTaskTrees) tasks [53]. The services to be integrated are discovered from the service registry and, for each one of them, a service behavioral model is automatically derived. This models is synthesized by eliciting the behavioral protocol of the services starting from the interface descriptions of the services, as discovered from the service registry. The adopted elicitation process extends the work in [9] and it is based on a combination of syntactic interface analysis and testing. The described CHOReOS approach can be seen as a particular instance of EAGLE. However, also in CHOReOS we do not consider the uncertainty and, at the same time, the accuracy of the elicited models as well as non-functional properties. This limitation prevents the producer from acquiring confidence in the final solution since there is no way to measure its "goodness".

5.2 Research Directions

The EAGLE vision asks for techniques to assist the producer in creating the appropriate integration means to compose the observed software together in order to produce an application that satisfies the goal. Architectural patterns and styles [60] and integration patterns [36] might be suitably exploited and instantiated to produce an integration architecture that interrelates models, which represent the software to be used to produce the desired software, with additional integrator models, which ease the collaboration and integration of the existing software to be used.

Moreover, formal methods should be exploited to automatically synthesizing integrator models. In particular we aim at synthesizing the extra-logic required to either cope with the uncertainty degree of the elicited models or implements additional logic expressed in the goal specification. The first case is when additional integration behaviour is synthesized to deal with interactions that can be performed by a service although they are not specified in its elicited model. The second concerns scenarios in which skeleton code is synthesized so to allow the software producer to complete it by only providing the functionality to be added without caring about possible side effects on the achievement of the goal, which is instead ensured by the synthesized integration code. Another strand concerns the satisfaction of non functional goals that requires the development of a whole set of compositional results on the integration means for non functional properties. First results in this direction might be found in [45].

6. MODEL INTEROPERABILITY AND CODE GENERATION

6.1 State of the Art

By shifting the focus of software development from coding to modelling, Model-Driven Engineering (MDE) refers to the systematic use of models as first class entities throughout the software engineering life cycle. The intention is to better manage the increasing complexity of modern systems while preserving the values of quality attributes obtained through the usual code-centric techniques. Coordinated collections of models and modelling languages are used to describe software systems on different abstraction layers and from different perspectives. In general, domains are analysed and engineered by means of a metamodel, i.e., a coherent set of interrelated concepts. A model is said to conform to a metamodel, or in other words it is expressed by the concepts encoded in the metamodel, constraints are expressed at the metalevel, and model transformations occurs when a source model is modified to produce a target model. In particular, model transformations play a central role since represent the glue between several levels of abstraction and enable the generation of different artefacts for documentation or analysis purposes, and even the generation of implementation code [13]. Model transformations refine and/or evolve a model into a different artefact: a new model (e.g., expressed in a different language), an abstraction of the original model, text (e.g., source code), or some other representation needed for a specific domain context [51].

The ability to synthesize artefacts from models helps ensuring the consistency between different interrelated artifacts. The automated transformation process is often re-

ferred to as "correct-by-construction", as opposed to conventional handcrafted "construct-by-correction" software development processes that are tedious and error prone [55]. At top level, model transformation approaches can be distinguished between model-to-model and model-to-text. The distinction is that, while a model-to-model transformation creates its target as a model which conforms to the target metamodel, the target of a model-to-text transformation essentially consists of strings.

Over the years model transformations have been applied for different purposes [23], such as to map and synchronize models at the same level or different levels of abstraction [19], to create different views on a system [12], and for model evolution tasks such as model refactoring [49, 59] and model differences [21]. Moreover, model transformations have been used in different application domains, e.g. to support the model driven development of Web applications and their evolution [20], the interoperability among various ADLs [44], the extensibility of ADLs [29], to manage the upgrade of Linux systems [26], and to automatically synthesize a choreography out of a specification of it and a set of services discovered as suitable participants [7].

6.2 Research Directions

Within EAGLE, a system is viewed as a community of interrelated models representing different viewpoints, different levels of abstraction, and even their different forms of integration. Hence, an adequate and systematic support to model interoperability is required possibly between the different modelling languages [19, 44, 35]. The idea is to conform, realise, implement, refine, or compose models forming a coordinated community of models, metamodels and, more specifically, domain-specific modelling languages. The disclosed challenge requires suitable correspondences (both functional and non functional) and fine-grained mappings which will be evaluated according to the semantics of the weaving associations specifically defined for the considered application domain [18].

Another research direction concerns the management of the evolution of model transformations, which may be changed in order to accommodate unforeseen requirements or to adapt the transformations with respect to changes operated on the corresponding metamodels (we call such a situation as metamodel/transformation co-evolution). The metamodel/model co-evolution problem has been already intensively explored and a research corpus is available (e.g., [27]). Most of the existing approaches provide tools and techniques to define and apply migration strategies able to take as input models conforming to the original metamodel and to produce as output models conforming to the evolved metamodel. On the contrary, the metamodel/transformation co-evolution problem is still open and requires further investigations. In fact, in addition to the *conformance* relation that must always hold between models and metamodels, adapting transformations has to take into account the *domain conformance* relation [28] between the definition of a transformation and its metamodels.

Finally, it is necessary to conceive code generators able to produce correct-by-construction code required to integrate the considered and observed third-party services. Dedicated techniques have to be developed to identify the parts of the generated code that cannot be modified by developers to not invalidate the goal and the architectural choices. The code generators will be configurable and extensible in order to do not limit its applicability to specific domains. In particular, depending on the target platform the code generator will provide the means to retrieve the proper already existing transformation templates to be applied for the generation, or to add new ones in case of platforms which have not been considered before.

7. CONCLUSION

This paper identifies key challenges that we believe will characterize and steer the research in software in the next future. Software is increasingly produced by reusing and integrating third-parties software according to a certain goal. Moreover, third-parties software is typically black-box and often provided without a machine readable documentation. Therefore, software of the next future asks to extract suitable observational models from third-parties software and devise appropriate integration means that permit to obtain the needed software system that satisfies the goal.

Coping with the identified challenges requires to put at work different expertises and skills together, hence asking for a multi-disciplinary research efforts in functional and non functional system modelling, specification mining, architecture and connector synthesis, model-driven development.

8. ACKNOWLEDGMENTS

This work is supported by the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreements: number 257178 (project CHOReOS - Large Scale Choreographies for the Future Internet - `www.choreos.eu`), and number 231167 (project CONNECT - Emergent Connectors for Eternal Software Intensive Networked Systems - `http://connect-forever.eu/`). We would like to thank also Vittorio Cortellessa for his contribution on non-functional aspects. Finally, we would like to thank Ivano Malavolta for providing support for the BusOnAir example (`http://www.busonair.eu/`).

9. REFERENCES

[1] G. Ammons, R. Bodík, and J. R. Larus. Mining specifications. In *Proceedings of the 29th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '02, pages 4–16, New York, NY, USA, 2002. ACM.

[2] M. Autili, V. Cortellessa, D. Di Ruscio, P. Inverardi, P. Pelliccione, and M. Tivoli. Eagle: engineering software in the ubiquitous globe by leveraging uncertainty. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, ESEC/FSE '11, pages 488–491, New York, NY, USA, 2011. ACM.

[3] M. Autili, V. Cortellessa, D. Di Ruscio, P. Inverardi, P. Pelliccione, and M. Tivoli. Integration architecture synthesis for taming uncertainty in the digital space. In *Proceedings of the 17th Monterey conference on Large-Scale Complex IT Systems: development, operation and management*, pages 118–131, Berlin, Heidelberg, 2012. Springer-Verlag.

[4] M. Autili, D. Di Ruscio, P. Inverardi, P. Pelliccione, and M. Tivoli. ModelLAND: where do models come

from. In *To appear*, Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013.

[5] M. Autili, P. Inverardi, and P. Pelliccione. Graphical scenarios for specifying temporal properties: an automated approach. *Automated Software Engg.*, 14(3):293–340, Sept. 2007.

[6] M. Autili, L. Mostarda, A. Navarra, and M. Tivoli. Synthesis of decentralized and concurrent adaptors for correctly assembling distributed component-based systems. *J. Syst. Softw.*, 81(12):2210–2236, Dec. 2008.

[7] M. Autili, D. Ruscio, A. Salle, P. Inverardi, and M. Tivoli. A model-based synthesis process for choreography realizability enforcement. In V. Cortellessa and D. Varró, editors, *Fundamental Approaches to Software Engineering*, volume 7793 of *Lecture Notes in Computer Science*, pages 37–52. Springer Berlin Heidelberg, 2013.

[8] T. Berg, B. Jonsson, and H. Raffelt. Regular inference for state machines using domains with equality tests. In *Proc. of FASE'08/ETAPS'08*, pages 317–331, 2008.

[9] A. Bertolino, P. Inverardi, P. Pelliccione, and M. Tivoli. Automatic synthesis of behavior protocols for composable web-services. In *Proc of ESEC/FSE '09*, 2009.

[10] A. W. Biermann and J. A. Feldman. On the synthesis of finite-state machines from samples of their behavior. *IEEE Trans. Comput.*, 21(6):592–597, June 1972.

[11] R. Bruni, I. Lanese, and U. Montanari. A basic algebra of stateless connectors. *Theor. Comput. Sci.*, 366(1):98–120, Nov. 2006.

[12] R. I. Bull, M.-A. Storey, J.-M. Favre, and M. Litoiu. An architecture to support model driven software visualization. In *Proceedings of the 14th IEEE International Conference on Program Comprehension*, ICPC '06, pages 100–106, Washington, DC, USA, 2006. IEEE Computer Society.

[13] J. Bézivin. On the unification power of models. *Software & Systems Modeling*, 4(2):171–188, 2005.

[14] K. L. Calvert and S. S. Lam. Formal methods for protocol conversion. *IEEE J.Sel. A. Commun.*, 8(1):127–142, Sept. 2006.

[15] C. Canal, P. Poizat, and G. Salaun. Model-based adaptation of behavioral mismatching components. *Software Engineering, IEEE Transactions on*, 34(4):546–563, 2008.

[16] S. Ceri, D. Braga, F. Corcoglioniti, M. Grossniklaus, and S. Vadacca. Search computing challenges and directions. In *Proc of ICOODB'10*, pages 1–5, 2010.

[17] L. Cheung, R. Roshandel, N. Medvidovic, and L. Golubchik. Early prediction of software component reliability. In *Proceedings of the 30th international conference on Software engineering*, ICSE '08, pages 111–120, New York, NY, USA, 2008. ACM.

[18] A. Cicchetti and D. Di Ruscio. Decoupling web application concerns through weaving operations. *Sci. Comput. Program.*, 70(1):62–86, Jan. 2008.

[19] A. Cicchetti, D. Di Ruscio, R. Eramo, and A. Pierantonio. Jtl: a bidirectional and change propagating transformation language. In *Proceedings of the Third international conference on Software language engineering*, SLE'10, pages 183–202, Berlin, Heidelberg, 2011. Springer-Verlag.

[20] A. Cicchetti, D. Di Ruscio, L. Iovino, and A. Pierantonio. Managing the evolution of data-intensive web applications by model-driven techniques. *Softw. Syst. Model.*, 12(1):53–83, Feb. 2013.

[21] A. Cicchetti, D. D. Ruscio, and A. Pierantonio. A metamodel independent approach to difference representation. *Journal of Object Technology*, 6(9):165–185, 2007.

[22] J. E. Cook and A. L. Wolf. Discovering models of software processes from event-based data. *ACM Trans. Softw. Eng. Methodol.*, 7(3):215–249, July 1998.

[23] K. Czarnecki and S. Helsen. Feature-based survey of model transformation approaches. *IBM Syst. J.*, 45(3):621–645, July 2006.

[24] V. Dallmeier, N. Knopp, C. Mallon, G. Fraser, S. Hack, and A. Zeller. Automatically generating test cases for specification mining. *IEEE Transactions on Software Engineering*, 38(2):243–257, 2012.

[25] C. Damas, B. Lambeau, P. Dupont, and A. van Lamsweerde. Generating annotated behavior models from end-user scenarios. *IEEE Trans. Softw. Eng.*, 31(12):1056–1073, Dec. 2005.

[26] R. Di Cosmo, D. Di Ruscio, P. Pelliccione, A. Pierantonio, and S. Zacchiroli. Supporting Software Evolution in Component-Based FOSS Systems. *Science of Computer Programming*, 76(12), 2011.

[27] D. Di Ruscio, L. Iovino, and A. Pierantonio. Coupled evolution in model-driven engineering. *IEEE Software*, 29(6):78–84, Nov. 2012.

[28] D. Di Ruscio, L. Iovino, and A. Pierantonio. A methodological approach for the coupled evolution of metamodels and atl transformations. In *6th International Conference on Model Transformation (ICMT2013)*, volume 7909 of *LNCS*, pages 60–75. Springer, 2013.

[29] D. Di Ruscio, I. Malavolta, H. Muccini, P. Pelliccione, and A. Pierantonio. Developing next generation adls through mde techniques. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, pages 85–94, New York, NY, USA, 2010. ACM.

[30] M. Dumas, M. Spork, and K. Wang. Adapt or perish: Algebra and visual notation for service interface adaptation. In S. Dustdar, J. Fiadeiro, and A. Sheth, editors, *Business Process Management*, volume 4102 of *Lecture Notes in Computer Science*, pages 65–80. Springer Berlin Heidelberg, 2006.

[31] Y. Ephraim and N. Merhav. Hidden markov processes. *IEEE Transactions on Information Theory*, 48:1518–1569.

[32] D. Garlan. Software engineering in an uncertain world. In *Proc. of FSE/SDP'10*, pages 125–128, 2010.

[33] C. Ghezzi, A. Mocci, and M. Monga. Synthesizing intensional behavior models by graph transformation. In *Proc. of ICSE '09*, pages 430–440, 2009.

[34] D. Giannakopoulou, Z. Rakamarić, and V. Raman. Symbolic learning of component interfaces. In *Proceedings of the 19th international conference on Static Analysis*, SAS'12, pages 248–264, Berlin, Heidelberg, 2012. Springer-Verlag.

[35] R. Hilliard, I. Malavolta, H. Muccini, and

P. Pelliccione. On the composition and reuse of viewpoints across architecture frameworks. In *Proceedings of the 2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture*, WICSA-ECSA '12, pages 131–140, Washington, DC, USA, 2012. IEEE Computer Society.

[36] G. Hohpe and B. WOOLF. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. A Martin Fowler signature book. Addison Wesley Professional, 2004.

[37] P. Inverardi, R. Spalazzese, and M. Tivoli. Application-layer connector synthesis. In M. Bernardo and V. Issarny, editors, *Formal Methods for Eternal Networked Software Systems*, volume 6659 of *Lecture Notes in Computer Science*, pages 148–190. Springer Berlin Heidelberg, 2011.

[38] P. Inverardi and M. Tivoli. Automatic synthesis of modular connectors via composition of protocol mediation patterns. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 3–12, Piscataway, NJ, USA, 2013. IEEE Press.

[39] F. Jiang, Y. Fan, and X. Zhang. Rule-based automatic generation of mediator patterns for service composition mismatches. In *Proceedings of the 2008 The 3rd International Conference on Grid and Pervasive Computing - Workshops*, GPC-WORKSHOPS '08, pages 3–8, Washington, DC, USA, 2008. IEEE Computer Society.

[40] S. S. Lam. Correction to 'protocol conversion'. *IEEE Trans. Softw. Eng.*, 14(9):1376–, Sept. 1988.

[41] X. Li, Y. Fan, J. Wang, L. Wang, and F. Jiang. A pattern-based approach to development of service mediators for protocol mediation. In *Proceedings of the Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008)*, WICSA '08, pages 137–146, Washington, DC, USA, 2008. IEEE Computer Society.

[42] D. Lo, L. Mariani, and M. Santoro. Learning extended fsa from software: An empirical assessment. *J. Syst. Softw.*, 85(9):2063–2076, Sept. 2012.

[43] D. Lorenzoli, L. Mariani, and M. Pezzè. Automatic generation of software behavioral models. In *Proc. of ICSE '08*, pages 501–510, 2008.

[44] I. Malavolta, H. Muccini, P. Pelliccione, and D. Tamburri. Providing architectural languages and tools interoperability through model transformation technologies. *IEEE Trans. Softw. Eng.*, 36(1):119–140, Jan. 2010.

[45] A. D. Marco, P. Inverardi, and R. Spalazzese. Synthesizing self-adaptive connectors meeting functional and performance concerns. In *SEAMS*, pages 133–142, 2013.

[46] R. Mateescu, P. Poizat, and G. Salaün. Adaptation of service protocols using process algebra and on-the-fly reduction techniques. In *Proceedings of the 6th International Conference on Service-Oriented Computing*, ICSOC '08, pages 84–99, Berlin, Heidelberg, 2008. Springer-Verlag.

[47] K. Meinke. Automated black-box testing of functional correctness using function approximation. In *Proc. of ISSTA '04*, pages 143–153, 2004.

[48] K. Mishra and K. Trivedi. Uncertainty propagation through software dependability models. In *Software Reliability Engineering (ISSRE), 2011 IEEE 22nd International Symposium on*, pages 80 –89, 29 2011-dec. 2 2011.

[49] N. Moha, V. Mahé, O. Barais, and J.-M. Jézéquel. Generic model refactorings. In *Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems*, MODELS '09, pages 628–643, Berlin, Heidelberg, 2009. Springer-Verlag.

[50] M. Neil, N. Fenton, and M. Tailor. Using bayesian networks to model expected and unexpected operational losses. *Risk Analysis*, 25(4):963–972, 2005.

[51] R. Paige and J. Gray. Guest editorial to the special issue on model transformation. *Software & Systems Modeling*, 12(1):85–87, 2013.

[52] R. Passerone, L. de Alfaro, T. Henzinger, and A. Sangiovanni-Vincentelli. Convertibility verification and converter synthesis: two faces of the same coin [ip block interfaces]. In *Computer Aided Design, 2002. ICCAD 2002. IEEE/ACM International Conference on*, pages 132–139, 2002.

[53] F. Paternò and C. Santoro. Preventing user errors by systematic analysis of deviations from the system task model. *International Journal of Human-Computer Studies*, 56(2):225 – 245, 2002.

[54] H. Raffelt, B. Steffen, T. Berg, and T. Margaria. Learnlib: a framework for extrapolating behavioral models. *Int. J. Softw. Tools Technol. Transf.*, 11(5):393–407, Oct. 2009.

[55] D. Schmidt. Guest editor's introduction: Model-driven engineering. *Computer*, 39(2):25–31, 2006.

[56] D. C. Schmidt, M. Stal, H. Rohnert, and F. Buschmann. *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects*. John Wiley & Sons, Inc., New York, NY, USA, 2nd edition, 2000.

[57] H. Singh, V. Cortellessa, B. Cukic, E. Gunel, and V. Bharadwaj. A bayesian approach to reliability prediction and assessment of component based systems. In *Proc. of ISSRE '01*, 2001.

[58] B. Spitznagel and D. Garlan. A compositional formalization of connector wrappers. In *Proceedings of the 25th International Conference on Software Engineering*, ICSE '03, pages 374–384, Washington, DC, USA, 2003. IEEE Computer Society.

[59] R. Tairas and J. Gray. Increasing clone maintenance support by unifying clone detection and refactoring activities. *Inf. Softw. Technol.*, 54(12):1297–1307, Dec. 2012.

[60] R. Taylor, N. Medvidovic, and E. Dashofy. *Software Architecture: Foundations, Theory, and Practice*. Wiley, 2009.

[61] M. Tivoli and P. Inverardi. Failure-free coordinators synthesis for component-based architectures. *Sci. Comput. Program.*, 71(3):181–212, May 2008.

[62] G. Tselentis and A. Galis. *Towards the Future Internet: Emerging Trends from European Research*. Stand alone Series. IOS Press, Incorporated, 2010.

[63] S. Uchitel, G. Brunet, and M. Chechik. Synthesis of partial behavior models from properties and scenarios.

IEEE Trans. Softw. Eng., 35:384–406, May 2009.

[64] R. Vaculín, R. Neruda, and K. Sycara. An agent for asymmetric process mediation in open environments. In *Proceedings of the 2008 AAMAS international conference on Service-oriented computing: agents, semantics, and engineering*, SOCASE'08, pages 104–117, Berlin, Heidelberg, 2008. Springer-Verlag.

[65] R. Vaculin and K. Sycara. Towards automatic mediation of owl-s process models. In *Web Services, 2007. ICWS 2007. IEEE International Conference on*, pages 1032–1039, 2007.

[66] A. van Lamsweerde. *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Wiley, 2009.

[67] N. Walkinshaw, B. Lambeau, C. Damas, K. Bogdanov, and P. Dupont. Stamina: a competition to encourage the development and assessment of software model inference techniques. *Empirical Software Engineering*, pages 1–34, 2012.

[68] A. Wasylkowski, A. Zeller, and C. Lindig. Detecting object usage anomalies. In *Proc. of ESEC-FSE '07*, 2007.

[69] R. W. White and R. A. Roth. *Exploratory Search: Beyond the Query-Response Paradigm*. Synthesis Lect. on ICRS. Morgan & Claypool Publishers, 2009.

[70] G. Wiederhold and M. Genesereth. The conceptual basis for mediation services. *IEEE Expert: Intelligent Systems and Their Applications*, 12(5):38–47, Sept. 1997.

[71] D. M. Yellin and R. E. Strom. Protocol specifications and component adaptors. *ACM Trans. Program. Lang. Syst.*, 19(2):292–333, Mar. 1997.