A computational logic-based approach to verification of IT systems

Marco Alberti\(^1\), Federico Chesani\(^2\), Marco Gavanelli\(^1\), Evelina Lamma\(^1\), Paola Mello\(^2\), Marco Montali\(^2\), Sergio Storari\(^1\), and Paolo Torroni\(^2\)

\(^1\) ENIIF, University of Ferrara
Via Saragat, 1 - 44100 Ferrara (Italy)
\(^2\) DEIS, University of Bologna
Viale Risorgimento, 2 - 40126 Bologna (Italy)

Main Topic. 1.2 (Governance - Positioning IT in support for compliance with regulatory and business requirements (IT Governance, SOX, Basel II, ...))

1 Problem statement

More and more business scenarios involve open systems, i.e., systems composed of interacting entities whose behaviour is not predictable in advance. The complexity of such systems increases over time, both in terms of number of interacting entities and of space of possible behaviours.

For those open systems whose behaviour is relevant to the business, it is a natural requirement to be able to (i) specify them, and to (ii) verify that the member behaviour in fact complies to the specification.

To specify such systems, a language is needed that can cover their possible behaviours, and express the features of such behaviours that are desirable in a given business scenario. The language to be used would benefit from formal semantics, that can identify compliant from non compliant behaviours in a non ambiguous way.

The verification of compliance is, in general, performed by means of automated procedures. A verification procedure will be much more valuable if it is formally proved correct, with respect to the formal semantics of the specification language. Moreover, it is desirable that the system behaviour be tested for compliance against the specification itself, rather than against an error-prone translation.

2 Solution

A promising answer to all these requirements is the choice of languages and systems based on Computational Logic (CL). CL-based languages are declarative and equipped with formal semantics; and they come with proof-procedures with proved correctness properties, such as soundness and completeness with respect to the declarative semantics. The declarative nature of CL-based specifications makes them suitable also for non-IT experts, who might have difficulties in understanding and using operational formalisms.
One such system is the Srxuu framework [d]. Srxuu is an abductive logic language inspired by Ung and Kolawski’s xuu [1] originally developed in the context of the EU-funded SOCS project (IST-2001-32530) [17] to specify agent interaction protocols and social semantics of agents communication languages. The Srxuu language provides features not commonly found in other CL-based languages, and yet useful for open system specifications, such as universally quantified variables and CLP [13] constraints, which allow, for example, for a simple treatment of deadlines. The behaviour of the entities involved in the system is described by means of events (the actual behaviour) and expectations (the desired behaviour). In particular:

- events (atoms with functor H) provide a description of what happened (first argument of the atom) and the time at which it happened (second argument).
- expectations are of two kinds: positive (atoms with functor E), which represent something expected to happen, and negative (atoms with functor EN), which represent something expected not to happen. Expectations, as well as events, have two arguments, representing description and time.

Srxuu-based specifications have a declarative semantics based on abductive logic. A specification in the Srxuu language is composed of two elements:

- a knowledge base (a logic program, i.e., a set of clauses of the form Head ← Body) which describes declaratively domain-specific knowledge;
- a set of integrity constraints, implications of the form Body → Head that relate the actual and the expected behaviour of the interacting entities.

We refer the reader to [5] for a complete description of the language; several examples can be found later in this paper.

SCIFF is also the name of the proof procedure used to verify system behaviours against Srxuu-based specifications; the proof procedure has been proved to be sound and complete under reasonable assumptions. The Srxuu proof procedure has been integrated in a software component, named SOCS-SI [3], equipped with a GUI interfaced to several agent and coordination platforms.

SOCS-SI’s operation is depicted in Fig. 1: given a specification in the Srxuu language and a narrative, called history, of the interaction being discussed,
SOCS-SI verifies whether the history is compliant to the specification, as defined by the declarative semantics of the SCIFF language.

The SCIFF framework has also been extended to deal with problems such as proof of protocol properties [4] and interoperability verification of a-priori compliance to choreographies [2] of web services.

3 Evidence that the solution works

In this section we demonstrate the SCIFF framework’s flexibility by showing its application in two very different scenarios, which nonetheless have in common their openness: one at a low level (the TCP protocol), and one at a high level (medical guidelines).

For both scenarios, we describe the SCIFF-based specification and discuss the experimental results on verification of compliance.

3.1 Application: Verification of the Opening Phase of the TCP Protocol

The Transmission Control Protocol [16] is one of the most known and used protocols for the transmission of data over an Internet Connection (over the IP protocol). It has been published in 1981, and since then several different implementations of the protocol stack have been proposed, developed and deeply tested.

Recently, with the advent of the “third generation” of mobile phones, the use of the TCP protocol has been adopted for supporting application protocols over wireless connections, from the core network of the telecommunication providers to the user terminals. Each phone maker has equipped its products with its own TCP implementation.

Since some details of the TCP protocol have not been completely specified, it can happen that different phones exhibit slightly different behaviours when connecting to the core networks of telecommunications providers. We have formalized the opening phase of the TCP protocol in the SCIFF language, and we have studied the logs of the connections, obtained from a mobile telecommunications service provider, with SOCS-SI. The main objective of the analysis was to (possibly) identify non-compliances between the behaviour of the peers (traced in the form of event logs) and the formalization (based on SCIFF) of the protocol. The provider was interested in knowing which communications were not compliant, and why.

We present here the ICs regarding the “three-way handshaking” open modality, that can be summarized as follows:

1. a peer $A$ sends to another peer $B$ a $syn$ segment;
2. $B$ replies by acknowledging (with a $ack$ segment) $A$’s $syn$ segment, and by sending a $syn$ segment;
3. $A$ acknowledges $B$’s $syn$ segment with a $ack$ segment, and starts sending data.
Specification 3.1 The Three-way Handshake opening phase of the TCP Protocol

\[ IC_1 : \]
\[
\begin{align*}
H & (tell(A, B, tcp(syn, null, NSynA, AckNumber), D), T1) \\
\rightarrow & E (tell(B, A, tcp(syn, ack, NSynB, NSynAAck), D), T2) \\
& \land NSynAAck = NSynA + 1 \land T2 > T1.
\end{align*}
\]

\[ IC_2 : \]
\[
\begin{align*}
H & (tell(A, B, tcp(syn, null, NSynA, AckNumber), D), T1) \\
& \land H (tell(B, A, tcp(syn, ack, NSynB, NSynAAck), D), T2) \\
& \land T2 > T1 \land NSynAAck = NSynA + 1 \\
\rightarrow & E (tell(A, B, tcp(null, ack, NSynAAck, NSynB Ack), D), T3) \\
& \land T3 > T2 \land NSynB Ack = NSynB + 1.
\end{align*}
\]

\[ IC_3 : \]
\[
\begin{align*}
H & (tell(A, B, tcp(syn, null, NSynA, ANY), D), T1) \\
& \land ta(TA) \\
\rightarrow & EN (tell(A, B, tcp(syn, null, NSynA, ANY), D), T2) \\
& \land T2 < T1 \land T2 > T1 − TA.
\end{align*}
\]

\[ SOKB : \]
\[
ta(1000msec).
\]

Specification 3.1 shows how the opening phase has been represented by means of the SCIff Language. In particular, \( IC_1 \) says that if \( A \) sends to \( B \) a \textit{syn} segment, whose sequence number is \( NSynA \), then \( B \) is expected to send to \( A \) an \textit{ack} segment, whose acknowledgment number is \( NSynA + 1 \), at a later time. Moreover (three way handshake), \( B \) is expected to send (within the same message) a \textit{syn} with another sequence number \( NSynB \).

\( IC_2 \) says that, if the previous two messages have been exchanged, then \( A \) is expected to send to \( B \) an \textit{ack} segment acknowledging \( B \)'s \textit{syn} segment, and with acknowledgement number is \( NSynB + 1 \), where \( NSynB \) is the sequence number of \( B \)'s \textit{syn}.

The opening phase (restricted to the three way handshake) would be completely specified by the integrity constraints \( IC_1 \) and \( IC_2 \). However, within the collaboration with a telecom provider, some domain experts explicitly required to focus our attention on a problem they had previously spotted. The TCP protocol definition [16] explicitly states that if a first \textit{syn} message has been sent and no \textit{ack} message has been received, it is allowed to repeat the initial \textit{syn} message. Unfortunately, the specification does not specify the minimum time interval between each transmission of a \textit{syn} message.
As a consequence, the following situation can happen: a fast peer $A$ send a $syn$ message to a slower peer $B$. $B$’s answer is delayed because its computational load is very high. Thus, $A$ starts to re-transmit the $syn$ message, causing problems to $B$ (typically, a denial of service). In order to verify this hypothesis, the integrity constraint $IC_3$ has been added.

Specification 3.1 has been used to check the compliance of the interaction between mobile phones and a central server, taking the history from a log file. Evidence has been found that, after an initial $syn$ message and no $ack$ received, different mobile phones retransmit a $syn$ with different timings (depending on different implementations). If the server does not answer rapidly enough, certain mobile phones repeat the $syn$ message causing a denial of service on the server side. The use of the $SCIFF$ tools to this scenario has provided three results:

1. it was proved on the logs that a behaviour of certain mobile phones was responsible for the server problems (indeed domain experts had already hypothesized the problem, but a formal proof was appreciated);
2. it identified which phones exhibited that particular behaviour, paving the way for elaborating different solutions;
3. it allowed the user to establish a minimum time interval between each $syn$ transmission; then this minimum time interval was used to define the Quality of Service (QoS) for the core network.

3.2 Application: $SCIFF$ for specifying and verifying careflow protocols

As described in [8], careflows focus on the behavioural aspects of medical work described in clinical practice guidelines. Careflow systems implement workflow concepts in the clinical domain, coordinating the execution of health care services performed by different health care professionals and structures.

As case study for exploiting the potentialities of our approach w.r.t. careflow protocols, we have chosen the cervical cancer screening guideline proposed by the sanitary organization of the Emilia Romagna region of Italy [7]. Cervical cancer is a disease in which malignant (cancer) cells form in the tissues of the cervix; the screening program proposes several tests in order to early detect and treat cervical cancer.

The careflow protocol is depicted in Fig. 2.

More specifically, we considered the regional specification of the cervical cancer screening and translated it to a set of $SCIFF$ integrity constraints. The translation was relatively straightforward.

For example, let us consider the beginning of the careflow, which states that “When the screening center invites a patient to be subject to a pap-test at a certain date, then it could be the case that either the patient comes at the chosen date for the inspection or that the patient communicates a refusal to the screening center. The invitation has to be sent with at least a month’s notice, and the patient should communicate the refusal within two weeks from the invitation”. It could be mapped to the following integrity constraint, where $Scr$ is
Fig. 2. Cervical cancer screening careflow

the screening center, *Pat* is the patient and *T_e* is the date chosen for the test. Times are expressed in days.

\[
H(invitation(Scr, Pat, T_e, ID), T_i) \\
\rightarrow E(examExec(papTest, Pat, ID), T_{xx}) \land T_{xx} > T_i + 31 \\
\lor E(tell(Pat, Scr, refusal, ID), T_r) \land T_r > T_i \land T_r < T_i + 15.
\]

(1)

A second integrity constraint is used to impose mutual exclusion between the two alternatives, i.e. to express that the visit should not take place if the patient has explicitly communicated a refusal:

\[
H(tell(Pat, Scr, refusal, ID), T_r) \\
\rightarrow EN(examExec(papTest, Pat, ID), T_e) \land T_e > T_r.
\]

(2)

We have then applied the SCiFF formalization of the cervical cancer screening to evaluate compliance of 1950 execution traces, contained in a real database.

Some incorrect behaviours have been randomly introduced in the database, in order to deeply test our approach. Each execution contains several events: from the minimum of one event (the invitation to take part to the screening followed by no response) to the maximum of 18 events (the whole careflow). 1091 logs resulted to be not conformant w.r.t. the formalization we have initially proposed. These results were analyzed by a screening expert which confirmed the conformant classifications and proposed some modifications on the careflow.
model to avoid false non conformant classifications: some particular cases (not allowed by the former careflow model) have been taken into account as conformant. The second verification round has finally showed that 64 executions are still not conformant: this result agrees indeed with the “wrong” logs artificially introduced in the database.

4 Current status and next steps

The declarative and operational semantics of the SCIFF framework are now established and implemented. Ongoing work on SCIFF-based specification and verification is devoted to a broader experimentation on real-world cases, and to the translation of textual and graphical specification languages into the SCIFF language [9].

A further application that we are currently addressing is process monitoring and mining, described in the following.

4.1 Application: business process monitoring and mining with SCIFF

Using SCIFF for conformance checking Besides a whole interaction protocol, the SCIFF language can be used to represent single properties of an interaction such as a business process.

In this way, SCIFF is used to classify logs of a business process as compliant or non compliant w.r.t. the specified property (or properties), operating similarly to the ProM LTL Checker [20]. The classification of execution traces could be useful for a business manager, who does not want (and is not able) to describe and verify the entire, procedural and complex workflow of his/her company, but aims to check whether some constraints or business rules are indeed satisfied during its execution. Hence, the purpose is to specify different requirements in a high-level, intuitive and declarative way and evaluate how many execution traces have satisfied them.

For example, we could check whether the process execution traces satisfy the 4-eyes principles, which states that two given activities should not be performed by the same person. By denoting such activities as $a$ and $b$, and by representing performed activities as atoms of the kind $\text{performed}(\text{Activity}, \text{Originator})$ the 4-eyes principle could be expressed in SCIFF as a denial, namely:

\[
\nonumber H(\text{performed}(a, O), T) \\
\land H(\text{performed}(b, O), T_2) \\
\rightarrow false.
\]

Learning SCIFF rules In recent years, many different proposals have been developed for mining process models from execution traces (e.g. [1, 18, 12]). All these approaches aim at discovering complex and procedural process models,
and differ by the common structural patterns they are able to mine. While recognizing the extreme importance of such approaches, we advocate the necessity of discovering also declarative knowledge, in the form of process fragments or business rules/policies, from execution traces.

By following this approach we do not mine a complete process model, but rather discover a set of common declarative patterns and constraints. Being declarative, this information captures what is the high-level process behaviour without expressing how it is procedurally executed, hence giving a concise and easily interpretable feedback to the business manager.

We have developed an approach for the automatic discovery of SCliff rules from a set of process execution traces, previously labeled as compliant or not [14]. Adopting a logic programming representation it is possible to exploit all the techniques developed in the field of Inductive Logic Programming (ILP for short) [15] for learning models from examples and background knowledge.

Among these techniques, we have applied a modified version of the ICL algorithm [10] to the problem of learning SCliff rules. Each rule is seen as a clause that must be true on all the positive traces and false on some negative ones. The theory composed of all the SCliff rules must be such that all the rules are true when considering a positive trace and at least one rule is false when considering a negative one.

**Implementation**

We are integrating both our mining algorithm and the SCliff conformance checker inside ProM 3 [20], an extensible framework which supports a plenty of plug-ins for process mining. The framework envisages process execution traces encoded in MXML, an XML-based extensible format capable to store event logs. The integration schema is shown in figure 3.

SCliff Checker is able to classify a set of execution traces w.r.t. a given SCliff property. Such a property (or set of properties) could be graphically configured by the user or mined from a MXML training set, by exploiting the SCliff Miner. SCliff Miner encapsulates the variant of ICL that learns SCliff rules. The plug-in takes as input an MXML log that contains traces previously classified as compliant or non compliant.

The classification pre-processing step could be done manually or by exploiting the SCliff checker. By adopting the second approach, the user could specify an high-level classification criterion (expressed in terms of a SCliff rule) to choose which traces have to be considered as correct or not.

5 Competitive approaches

A number of general approaches to formal specification and verification can be found in the literature (such as those based on finite state machines, model
checking, workflow managements etc.). Due to lack of space, we limit to those based on logic, which are closer to ours.

In [22], a variant of the Event Calculus is applied to commitment-based protocol specification. The semantics of messages (i.e., their effect on commitments) is described by a set of operations whose semantics, in turn, is described by predicates on events and fluents; in addition, commitments can evolve, independently of communicative acts, in relation to events and fluents as prescribed by a set of postulates.

Artikis et al. [6] present a theoretical framework for providing executable specifications of particular kinds of multi-agent systems, called open computational societies, and present a formal framework for specifying, animating and ultimately reasoning about and verifying the properties of systems where the behaviour of the members and their interactions cannot be predicted in advance. Such specifications are based on and motivated by the formal study of legal and social systems, and operators of Deontic Logic [21] are used for expressing legal agent behaviour. Differently from [6] (and from other work on normative systems), we do not explicitly represent concepts such as institutional power of the society members and validity of action. Instead, permitted are all events that do not cause a violation.

Van der Aalst et al.[19] propose a language based on LTL to express properties of processes, and a checker to verify whether a business process log satisfies them. This checker has been integrated in the ProM framework [20]. The perspec-
tive of this work is similar to ours, in that we check process logs for compliance against a whole interaction protocol or individual properties; however, the different logic (abductive logic combined with constraints) that we use to specify protocols or properties let us express, for instance, time deadlines.

Acknowledgements

This work has been partially supported by the MIUR PRIN 2005 projects Specification and verification of agent interaction protocols and Vincoli e preferenze come formalismo unificante per l’analisi di sistemi informatici e la soluzione di problemi reali, and by the MIUR FIRB project Tecnologie Orientate alla Conoscenza per Aggregazioni di Imprese in Internet.

References