Sensoria

016004
Software Engineering for Service-Oriented Overlay Computers

D3.3.a: An Overview of Techniques for Behavioural Properties

Lead contractor for deliverable: ISTI-CNR
Author(s): Luís Caires (FFCUL), Artur Zawłocki(WARSAW), Andrea Corradini (PISA), Franco Mazzanti (ISTI), Michele Loreti (UNIFI), Hanne Riis Nielson (DTU)

Due date of deliverable: August 31, 2006
Actual submission date: August 31, 2006
Revision: Final
Dissemination level: PU

Contract start date: September 1, 2005 Duration: 48 months
Project coordinator: LMU
Partners: LMU, UNITN, ULEICES, UWARSAW, DTU,
PISA, DSIUF, UNIBO, ISTI, FFCUL, UEDIN, ATX,
TILab, FAST, BUTE, S&N, LSS-Imperial, LSS-UCL

Information Society
Technologies

Integrated Project funded by the
European Community under the
“Information Society Technologies”
Programme (2002—2006)

Executive Summary

The main goal of WP3 is to coordinate the partners efforts towards the development of qualitative analysis methods for global services. In particular, this deliverable reports on the research activity carried on during the first 12 months by the SENSORIA partners in the advancement of the state of art for a wide spectrum of techniques suitable for the description and analysis of the behavioral properties of services.

Twelve original contributions have emerged as result of this research activity. In the following sections of this document an overview is given of the performed activity and the results are described with a limited level of technical details . The full details of the contributions can be found in the twelve papers mentioned in the final ”Relevant Sensoria Publications and Reports” Section.

Contents

1 Introduction

The main goal of WP3 is to coordinate the partners efforts towards the development of qualitative analysis methods for global services. While security and trust related issues are specifically analyzed in Task 3.1 (and presented in Deliverable 3.1.a), and issues related to the migration (or mobility) of clients and resources are analyzed in Task 3.2 (and presented in Deliverable 3.2.a), Task 3.3 is oriented to the development of techniques for describing and analyzing the dynamic behavioral properties of distributed services.

In this first period of the project (months 1-12) the various partners efforts have been concentrated in the advancement of the state of the art for a wide spectrum of techniques suitable for the description and analysis of the behavioral properties of services.

Temporal logics, and modal logics (which allow to mix temporal and other structural/spatial operators), are two of the promising techniques which have been investigated in several directions by several partners. The SENSORIA advancements in this area are presented in Sections 2,3,4,and 5. The design and experimentation with these logics is often being backed up with the development of prototypes allowing to experiment the verification of logical properties. Some preliminary results in this sense are already referenced in this report; further results will be reported as part of the activity of Task 3.4 and Task 3.5

Flow analysis, and type based analysis are representatives of a different approach more aimed to the static enforcement of correctness (sometimes maybe w.r.t. to approximations of the system under analysis) of distributed services rather than to the dynamic verification of them. The ongoing work on these techniques is presented in Sections 6 and 7.

Graph Transformation Systems are a flexible formalism for the specification of the operational behaviour of complex systems. Some activity has been done on the theoretical foundations of this technique with the plan of applying it for the modeling of a subset of the SENSORIA Core Calculus. This approach is presented in Section 8.

The activity done so far actually spawns in several directions and covers a wide range of techniques. This is quite natural for this first period of the project where the partners activities are more oriented to the analysis of the fitness of the various techniques in which they have greater expertise with respect to the class of problems in which SENSORIA is interested. For the next SENSORIA period (months 13-30), as one or more common SENSORIA languages (Core Calculus) will become stably defined it is expected they will provide an attractive playground for the investigated techniques allowing to achieve a greater and common focus with respect to SENSORIA goals for the qualitative evaluation of systems.

2 Spatial Logic based Analysis of Distributed Systems

The overall framework

Logical characterizations of concurrent behaviors have been introduced for a long time now. A basic result in the field, due to Hennessy and Milner, is the characterization of behavioral equivalence in process algebras as indistinguishability with respect to a modal logic. Such results are important not only theoretically, but also because of their influence in the design of practical specification languages for software systems. HML characterizes behavioral equivalence in the sense that two processes are strongly bisimilar if and only if they satisfy exactly the same formulas.

More recently, Spatial Logics for Concurrency have been proposed with the aim of specifying distributed behavior and other essential aspects of distributed computing systems. In general terms, these developments reflect a shift of focus in concurrency research, that has been building up from the last decade on, from the study of centralized concurrent systems to the study of general distributed systems. While centralized processes may be accurately modeled as pure objects of behavior, in distributed systems, and in particular service based systems, many other interesting phenomena besides pure interaction, such as location dependent behavior, and resource usage policies, must be also considered.

In technical terms, while more traditional modal logics for concurrency characterize pure behaviors, usually modeled by labeled transition systems, or synchronization trees, spatial logics aim to characterize the dynamics of computing systems with a rich spatial structure. Such systems have been usually modeled by a process calculus where some constructors are understood as spatial constructors. Such spatial constructors may give rise to appropriate spatial observations in the logic, enabling a spatial logic to talk about behavior in space. Another important ingredient of spatial logics, highlighted in other structural logics such as Reynolds-O'Hearn separation logic, is their resource awareness in the sense that spatial operators are able to separate, count, and identify resources; this sometimes seems to add an intentional character to these logics.

The work done

In this stream we have been following a few parallel research directions.

A first line of research concerns the use of spatial logics to express properties of rich type systems, in which, in addition to behavioral properties akin to those enforced in related type systems, such as session types, we may also be able to express resource usage patterns, dynamic ownership of resources, and independence. The formal definition and analysis of sophisticated type systems for concurrent or distributed process models tend to get, to our opinion, involved and sometimes a bit ad-hoc when properties such as those mentioned above are in the scope of the intended type system. This observation suggests that the purely syntactic approach to show soundness of typing, as initially advocated by Felleisen and Wright, may not scale so well when fairly complex computational models are involved, because of the consequent increase in complexity of the global invariants required by a classical subject reduction proof for such models. We are therefore investigating how spatial logics may be used to organize the combination of semantic and syntactic techniques both in the definition of types and in the construction of compositional proofs of soundness. We already have obtained promising results, in the form of a new type system for service based systems, able to enforce resource usage policies, and dynamic ownership of service references [2005-FMCO]. For this type system we have been able to provide a soundness proof based on semantic reasoning, where types are interpreted as logical properties (expressed in a spatial logic), and the intended safety properties are expressed as a logical predicate, in the spirit of the logical relations technique. A detailed publication describing the work carried out in this are is expected to be finished before month 12.

The second research stream concerns the development of tools and proof techniques for logics whose expressive power builds on the combination of behavioral and spatial observations. Here, we have been developing model checking algorithms and tools appropriate for distributed systems expressed in a version of the pi-calculus against an expressive spatial logic. The main output of this activity is the Spatial Logic Model Checker [2005-SLMC], which has been under development for a couple of years now and of which a new version was recently released. The process language now considered is the polyadic pi-calculus with choice and parametric recursion, while the logic is a rich spatial logic including recursively defined properties. The logic can be seen as an extension of the MPW logic developed for the pi-calculus, of which a model-checker is included in the Mobility Workbench Tool suite. However, in our case, the logic is also able to talk about system structure, hidden names, and dynamic evolution of spatial structure. An example of the source language accepted by our tool is shown below. The example concerns a token passing system where five server nodes are connected in a ring. A node may either pass the token to its neighbor, or remove itself from the ring. For this system we may automatically prove many properties. For example, the query:

"check System $\models$ always ring;"

establishes that the system always keeps a ring structure. The property of "being a ring" is defined as shown below, using a combination of spatial modalities and recursion.

```
 /* TOKEN RING SYSTEM */

defproc Exit(inCh,outCh) =
        select {[outCh=inCh].0; outCh!(inCh).0};

defproc IdleNode(inCh,outCh) =
        inCh?(newInCh).TokenOwner(newInCh,outCh)

and TokenOwner(inCh,outCh) =
        select {
             tau.Exit(inCh,outCh);
             outCh!(outCh).IdleNode(inCh,outCh)};

defproc System =
        (new l1,l2,l3,l4,l5 in
              (IdleNode(l1,l2) |
              IdleNode(l2,l3) |
              IdleNode(l3,l4) |
              IdleNode(l4,l5) |
              TokenOwner(l5,l1)));

/* PROPERTIES */

defprop exiting(inl,outl) =
        1 and (((inl != outl) or <>0) and < outl!(inl) > 0);

defprop node(inl,outl) =
        1 and
        (exiting(inl,outl) or
        (maxfix X(inLnk).
                ((< inLnk?(newInLnk) > X(newInLnk))
               or ((<> exiting(inLnk,outl))
                    and (< outl!(outl) > X(inLnk)))))
        (inl));

defprop ring =
        0 or
        (hidden lnk.
        (minfix Y(x).
              (node(x,lnk) or
              (hidden y. (node(x,y) | Y(y)))))
        (lnk));

check System |= always ring;
```

We have also been investigating logical characterizations of distributed behaviour [2006-DEQ]. In particular, we have addressed the tensions between intensionality and extensionality of spatial observations in distributed systems, having shown that there are natural models where extensional observational equivalences, defined from standard barbed congruence, may be characterized (in the same sense that bisimilarity is characterized by Hennessy-Milner Logic) by spatial logics including the composition and void operators. Our results support the claim that spatial observations, perhaps surprisingly, do not need to be always considered intensional, even if expressive enough to talk about the structure of systems. We expect to extend our results in order to support reasoning and analysis of richer computational models, in particular service based systems.

Attached papers

[2005-SLMC] Hugo Vieira, Luís Caires and Ruben Viegas; *The Spatial Logic Model Checker User's Manual v1.0*, Technical Report of Departamento de Informatica, FCT/UNL. November 2005.
Short abstract The Spatial Logic Model Checker is a tool allowing the user to automatically verify behavioral and spatial properties of distributed and concurrent systems expressed in a pi-calculus. The algorithm implemented (currently using on-the-fly model-checking techniques) is provably correct for all processes, and complete for the class of bounded processes [2], an abstract class of processes that includes the finite control fragment of the pi-calculus. The tool itself is written in ocaml, and runs on any platform supported by the ocaml distribution.

[2006-DEQ] Luís Caires and Hugo T. Vieira. *Extensionality of Spatial Observations in Distributed Systems*, proc. 13th Int. Workshop on Expressiveness in Concurrency (EXPRESS'06). June 2006.
Short abstract: We discuss the tensions between intensionality and extensionality of spatial observations in distributed systems, showing that there are natural models where extensional observational equivalences may be characterized by spatial logics, including the composition and void operators. Our results support the claim that spatial observations do not need to be always considered intensional, even if expressive enough to talk about the structure of systems. For simplicity, our technical development is based on a minimalist process calculus, that already captures the main features of distributed systems, namely local synchronous communication, local computation, asynchronous remote communication, and partial failures.

Sensoria integration and future plans.

Further extensions to the Spatial Logic Model Checker are foreseen, building on introducing extensions to HD automata to cope with spatial structure, in cooperation with UNIPI (Gianluigi Ferrari, Ugo Montanari). Concerning the second line of work, we would like to use our results to inspire new logics for service-based systems. A particular concern here is not only in expressiveness issues, but also in the motivation of new ways of expressing and specifying distributed behavior and location aware computation. With regard to the work on spatial type systems, we would like to investigate the relations between our approach and others being developed in SENSORIA relative to the typing resource usage, in particular the work of Massimo Bartoletti, Pier-Paolo Degano and Gianluigi Ferrari (UNIPI), in particular in what sense our techniques may be used to obtain compositionality of typing, something which seems required in the context of open-endedness present in service based systems. We would also like to consider how resource control policies, modeled by spatial usage protocols, could also be imposed through the use of spatial types in the Service Core Calculus (SCC) being developed within SENSORIA.

3 A Categorical Formalism for Specification of Distributed Component Systems

The overall framework

Our aim within Task 3.3 is to develop a formalism, consisting of a specification logic and a semantic model, for specification of systems built from distributed components, in which both structural and behavioural properties can be expressed. Ultimately, the system should be presented as a so called *institution* (the notion of an institution is a formalisation of the concept of logical system in the language of category theory, due to Burstall and Goguen) which would allow us to reuse well-established institution-independent tools for constructing specifications and implementations in a modular way.

Service-Oriented Architectures can be viewed as distributed systems of loosely coupled components, running concurrently and coordinating their activities. Compared to traditional component systems, in service-oriented architectures more emphasis is put on dynamic *system reconfiguration*, due e.g. to the discovery of new services or failures of the previously existing ones. A formalism for specification of service-oriented systems should therefore allow for describing the evolution of the system's structure in time: addition of new components, removal of the old ones, changes in component interconnections.

The most natural choice is to use the first-order logic for description of the system's configuration. One can then quantify over components and express properties such like:

> Every session component in the system is associated with exactly one service provider component and exactly one client component.

Adding the temporal aspect allows one to express properties of joint component execution:

> Service requests are handled in the FIFO order

or concerning component reconfiguration:

> Sending a request involves creating a new message component. The channel component connecting the client and the service is destroyed after the session is closed.

There exist several specification formalism based on the first-order temporal logic. Typically, the so called "state-as-algebra" approach is adopted. That is, a component system is represented by a transition system with states labelled with algebras (or first-order structures) representing the system's structure. Transitions may be labelled with mappings or relations that allow one to keep track of the identities of system components. Dynamic reconfiguration can be represented in terms of transitions between global system states.

A crucial feature of any specification formalism is the possibility of building complex specifications (as well as implementations, in case the formalism in question can also describe how an implementation of a specification is built) in an incremental, modular way. In this respect, adopting the global, monolithic system view, as in the state-as-algebra formalisms, does not seem to be a good choice. Instead, we propose to represent a component system as a collection of transition system models of individual components, together with relations between those models that represent component interactions. Technically, a model of a system is a diagram in an appropriate category of component models. Such models, or implementations, can be built incrementally, by adding more components that depend on the previously existing ones.

Previously existing formalisms with categorical semantics, such as Fiadeiro's CommUnity, allow one to specify systems with a fixed structure. Moreover they lack the desirable features of first-order logic, such as quantification or counting of components (the same can be said of various spatio-temporal logics based on process-algebras).

The work done

The formalism we propose is equipped with a categorical semantics. In [2006-Zawa], we start with a particular category of component models. Roughly, the models are labelled transition systems extended in a straightforward way with state-dependent attributes. For such models a suitable modal logic is proposed. Formulae of this logic describe behaviour of individual components.

A well-known categorical principle is to represent complex systems as *categorical diagrams* in a suitable category of component models. Behaviour of a system can be then represented as a limit object of the corresponding diagram, i.e. a categorical construction that produces a minimal component model having all the components of the diagram as subcomponents. Component interconnection can be represented by sharing of common subcomponents. In [2006-Zawa] we propose to represent complex systems as categorical diagrams of previously defined component models. The component logic is extended accordingly: to the modal logic for individual components we add several new syntactic constructs for expressing the structure of the system. In particular, we add the possibility of quantifying over components and binding them to variables, thus introducing elements of the first-order logic. In the resulting logic, temporal modalities and structural operators can be freely interleaved.

The semantics of the logic is defined in such a way, that formulae are evaluated in various sub-diagrams of the whole model diagram. Such sub-diagrams represent many alternative component configurations – there may be no single ”universal” configuration involving all possible components, but still one can describe various local configurations, involving only some of the components. Moreover, new components may appear in a configuration and the old ones may terminate. This allows us to model system reconfiguration.

We show that the formalisms inherits the expressive power of the first-order temporal logic. In particular, this means that the logic is incomplete, i.e., the set of valid formulae is not recursively enumerable.

In [2006-Zawb] we generalize the semantics somewhat by using a more general notion of component models. An important contribution of the paper is the formalisation of the specification logic as an institution. This opens the possibility to consider structured and architectural specifications of service-oriented systems.

Attached papers

[2006-Zawa] A. Zawłocki, *Diagram Models for Interacting Components*, Submitted. Extended abstract presented on 3rd International Workshop on Formal Aspects of Component Software, FACS'06, Prague, Czech Republic, September 20-22, 2006.
short abstract: We present a semantic model and a logic for systems of concurrent components. Following the categorical approach, we define a category of component models in which limits can be used to construct systems from simpler components. A novel idea is to use diagrams, not just limit objects, as models of component systems. We also propose a logic that allows one to specify both behavioural and structural aspects of systems: temporal operators and structural predicates can be freely interleaved. Finally, we present an example specification of a system of interacting components.

[2006-Zawb] A. Zawłocki, *An Institution for Interacting Components*. Submitted. Extended abstract presented on 18th International Workshop on Algebraic Development Techniques, WADT'06, La Roche en Ardenne, Belgium, 1-3 June 2006.
short abstract: In this paper we propose a formalism for the specification of systems built from interacting components. We define a notion of a system model consisting of a categorical diagram of models for individual components. For such diagram models we propose a logic in which it is possible to specify both the temporal and the structural aspects of component configurations by freely interleaving temporal operators of the branching-time temporal logic with the constructs from the first-order logic. In particular, it is possible to describe dynamic system reconfiguration, for instance the creation or termination of system components. We formalise the diagram logic as an institution and provide an example specification.

Sensoria integration and future plans.

Our formalism can be used to provide an alternative model-theoretic semantics for service ontology developed within Task 1.1 (Service Description), and service composition, developed within Task 1.2 (Service Composition). Moreover, the formalism we develop can be specialized to various notions of services, in principle it should be applicable to variants of the core calculus emerging from WP2. Finally, by presenting our formalism as an institution we can use institution-independent refinement and verification techniques for specifications, which is relevant for Task 3.4 (Refinement and Verification Techniques).

In months 13–30 we plan to continue the work on foundational aspects of component system specification and develop some proof methods for our logic (in spite of the incompleteness result). Finally, we plan to develop some larger examples of specifications of service-oriented systems, using our formalism.

4 An Experimentation with State and Event based Temporal Logics

The overall framework

Most of the specification languages used to describe the dynamic behaviour of systems are either state-based or event-based. In the first case the description of a system is centered around the internal properties of the states; i.e. a system is typically seen as an evolving set of global variables or object attributes. In the second case the description of system is centered around the events occurring while moving from one state to next, which typically describe the interactions and the communications occurring among the system components and the external world.

When coming to the issue of describing the behavioral properties a system, this dichotomy is usually reflected into the kind of logic in which the behavioral properties are expressed. From one side we have state-based logics (CTL [1981-CE], CTL* [1986-EH], PTL [1980-PTL]), in which the basic predicates of the logic are propositions related to the labelling of the states, typically expressing the fact that in a state a certain basic property holds (e.g. an attribute or variable has a certain value). From the other side we have event-based logics (ACTL, ACTL* [1990-DNV] , HML[1980-HM]), in which the basic constructs are expressions related to the labelling of the evolutions occurring between two states, typically expressing the fact the during an evolution a certain event occurs (e.g. a synchronization between two components, or the emitting of a signal).

Indeed both paradigms are important for the specification of a real system and recent object oriented modelling languages allow to naturally describe in detail both aspects of the behaviour of a system. For these kinds of models, a logic which allows to directly express both state-based and event-based properties of the temporal behavior of a system is the most natural choice. Unfortunately most (if not all) of the state of the art verification environments take one or the other of the two described modelling approaches, failing to allow a comprehensive natural description of the system and their properties.

The work done

Our starting point is the full mu-calculus [1983-K], which is a powerful logic introduced by Kozen in 1983, which is known to subsume all the other both state based and event based logics, both in their linear time and branching time flavour. The problems with full mu-calculus, however are of two kinds. First, the complexity of the evaluation of a generic formula is exponential w.r.t. the size of the model, and this tends to quickly cause the intractability of the evaluation. Second, the intuitive meaning of a formula may become particularly hard to understand even for formulas with a few levels of recursion, and this makes the logic unattractive from user friendliness point of view. For these reasons we have seen a proliferation of many different logics which allowed from one side an efficient (in many cases also linear) evaluation complexity, and from the other side more intuitive higher level logical operators which hide the basic fixpoint primitives. The structure of the logic mu-UCTL [2005-GMaica] is essentially the mu-calculus (in its original both modal and propositional aspects) extended with the well known higher level operators like the Eventually, Always, Until operators (in the CTL / ACTL style). Moreover mu-UCTL allows a detailed specification of both the basic properties that states or system evolutions should satisfy due to its rich action and basic proposition expressions.

Experimentation with the logic mu-UCTL is being done in the context of the UMC [2006-MUG33] prototype. UMC, with its UML-like modelling language and its state-event based logic is an experiment in designing a logic, a language, and a verification system, allowing to easily express and verify both the internal details of system configurations and the details of the events occurring during the system evolutions. The evolution of the UMC prototype inside SENSORIA has been guided by the goal to support the potential features needed for the verification of service-oriented systems. Two new major versions of UMC have been produced (version 3.2 and 3.3), leading to a richer logic, a more powerful model description language and to an improved user interface. ISTI is in the process of evaluating the applicability of UMC for one of the suggested examples of the Telecom case Study (in particular in the Asynchronous SOAP protocol definition [2006-BGMM]). The improvement of the possible interactions / integrations of UMC with other tools used inside SENSORIA (as for example Hugo), is also being another relevant direction of work. Currently UMC still has the form of a fastly evolving running prototype, mainly tailored to the experimentation of verification/modelling ideas in the field of service-oriented computing, and has not undergone the major effort needed in terms of stability/documentation/testing/user-support required for an official public release.

Without showing all the details of the mu-UCTL grammar we will show here a few examples of properties which can be described in our logics; in doing that we take as reference model a system composed by a set of named active objects (obj1, obj2,...), where each object holds a set of local private attributes, and where objects communicate through the sending of asynchronous signals or by performing synchronous operation calls.

”For all execution paths in which no ”failure” signals are emitted by the system, we will eventually reach a final status where the attribute ”status” of object ”main” will be the token ”done”.

```
min Z: ((FINAL and main.status=done) or
         (not FINAL and [not failure] Z) )
```

”It cannot happen that obj2 sends a ”response” signal to obj1, without having previously received a ”request” signal from it.”

```
not E[ true {not obj1:obj2.request} U <obj2:obj1.response> true ]
```

”It cannot happen that obj1 sends a signal to obj2, before without having first received obj2 from as a result of an operation call to obj3”.

```
not E [ true {not obj3:obj1.return(obj2)} U <obj1:obj2.*> true ]
```

”As long as attribute x of obj1 is greater then 0, obj1 does not send signals or calls to obj2”.

```
not EF (obj1.x>0 and <obj1:obj2.*> true)
```

Attached papers

[2005-GMaica] S.Gnesi and F. Mazzanti *A Model Checking Verification Environment for UML Statecharts* Proceedings XLIII AICA Annual Conference, University of Udine - October 2005, AICA.
short abstract: In this paper we present the state/event-based temporal logic mu-UCTL which is a logic oriented towards a natural description of dynamic properties of UML models. This logic allows to specify the basic properties that a runtime system configuration should satisfy and to combine these basic predicates with logic and temporal operators which allow to take into consideration also the events performed by the system when evolving from one system configuration to another. Doubly Labelled Transition Systems are the semantic domain for mu-UCTL. The logic is supported by a prototypical verification environment under development at ISTI built around the ”on the fly” UMC model checker.

[2006-BGMM] M.H. ter Beek and S. Gnesi and F. Mazzanti and C. Moiso, *Formal Modelling and Verification of an Asynchronous Extension of SOAP*. ISTI Technical Report ISTI-2006-TR-19, June 2006
short abstract: Current web services are largely based on a synchronous request-response model that uses the Simple Object Access Protocol SOAP. Next-generation telecommunication networks, on the

contrary, are characterized by the need to handle asynchronous interactions among distributed service components, e.g., to deal with events produced by the network resources. As these worlds are more and more converging into a single application context, several solutions have been proposed to deal with asynchronous events in the context of web services. In this paper we formalize and verify one such approach, viz., an original asynchronous extension of SOAP, and draw some conclusions. The formal model is specified as a set of communicating state machines. The semantics of the model is seen as a doubly-labelled transition system, and its behavioural properties are expressed in the action- and state-based temporal logic mu-UCTL and verified with the on-the-fly model checker UMC.

[2006-MUG33] F.Mazzanti *UMC 3.3 User Guide* ISTI Technical Report 2006-TR-33, September 2006.
short abstract: In this report we illustrate the version 3.3 of UMC, a prototype tool for the exploration, analysis and on-the-fly model checking of the dynamic behaviour of UML-like models. Models are described as collections of communicating objects. Objects belong to classes, whose dynamic behaviour is described by statecharts. The logic supported by the tool is an extension of mu-ACTL and has the power of full mu-calculus. The modelling language, the supported logic, the verification approach, and the user interface are all described in detail.

Sensoria integration and future plans.

State and Event based logics seem well fitting the needs for the specification of the behavioral properties of service-oriented systems, and in particular for the analysis of communication protocols followed during the service interactions, or for describing the intended behavior of a system at different levels of abstraction. We can indeed imagine the analysis of a system one time occurring at low level and inspecting the evolution of some specific object attribute during the lifetime of the system, and another time occurring at a much higher lever inspecting for example just the system interaction with the external world. Being able to perform these two kinds of analysis in the same uniform logical framework and with the same tool is certainly a positive fact.

In the next 13-30 months we plan to use UMC for the analysis of one of telecom case studies, further adapting the current prototype according to need that will arise as a result of the analysis. We also plan to use the experience gained in the definition of the logics and its verification algorithm for evaluating their applicability to the SENSORIA core calculi. Finally we will to take into consideration the issues arising about a smoother integration of our prototypes with the other developed SENSORIA tools and prototypes.

5 MoMo: A Modal Logic for Mobility

The overall framework

A well established and successful approach to modelling and verifying properties of concurrent systems is the one based on process algebras and temporal logics: concurrent systems are specified as terms of a process algebra while properties are specified as temporal logic formulae. Labelled transition systems are associated to terms via a set of structural operational semantics rules and model checking is used to determine whether the transition system associated to the term enjoys the property specified by the temporal formula.

The success of the approach has stimulated research on these general models and has permitted the development of general tools and results for minimization, animation, and axiomatizations and for equivalence and model checking.

In the last decade, stimulated by new applications of network aware programming, several new formalisms based on process algebras but with new constructs for modelling network topology, name passing, resource usage and mobility have been proposed.

The new primitives of these distributed nominal calculi, where together with actions, names and resources play a central role, render it more difficult to use labelled transition systems and classical operational semantics. To model distinguishing features of these calculi, like *name scoping* or *process and data distribution* , transition labels are enriched to consider not only the actions a system can perform but also the information about the state of the system. A disadvantage of this approach is that it needs to be tailored on the specific calculus and cannot be easily generalized.

Each distributed nominal calculus defines its own structured set of transition labels and this does not permit considering a common semantic framework that can be used as the base for defining common tools, like spatial and temporal logics, for reasoning about systems specified using the different calculi.

The work done

To provide a general framework for describing the semantics of mobile calculi we introduce a variant of LTS that we call *Multiple Labelled Transition Systems* (MLTS) [2006 -DNL] [2005-DL04] because they are equipped with different transition relations that capture different aspects of system behaviour namely *actions execution,resources creation and consumption* and *name revelation*.

An MLTS consists of a set of *states*, a set of *resources*, a set of *transition labels* and a *naming structure*. States describe system configurations, the naming structure permits associating public (known by the environment) names to each state. Resources model data (e.g. the values exchanged over a channel), computational environments (e.g. the locations where processes can be executed) or network links that can be used for interaction.

The interactions with the environment are modelled by means of three different transition relations: *computation relation*, that describes the interaction with the environment; *resource relation* that describes resource usage; and *revelation relation* that describes the names revealed to the environment.

For modelling properties of MLTS, we have introduced a temporal logic that consists of a number of basic operators to be used to describe specific properties/behaviours of mobile and distributed systems. Thus, together with the usual logical connectives and the operators for minimal and maximal fixed points, we have operators for describing dynamic behaviours (*temporal properties*), for modelling resource management (*state properties*), for keeping into account name handling (*nominal properties*), and for controlling mobile processes (*mobility properties*).

To show the usefulness of our proposal we illustrate how MLTS can be used to describe the operational semantics of two formalisms with opposite objectives, namely KKlaim and the asynchronous π-calculus. The former is a simplified version of Klaim. Klaim is an experimental programming language that supports a programming paradigm where both processes and data can be moved across different computing environments and relies on the use of explicit localities. The latter one is the generally recognized minimal common denominator of calculi for mobility.

Attached papers

[2006 -DNL] R. De Nicola and M. Loreti, *Multi Labelled Transition Systems for Nominal Calculi and their Logics* (Submitted).

short abstract: To provide a general framework for describing the semantics of mobile calculi we introduce Multi Labelled Transition Systems (LTS). These are generalization of LTS to deal with notions of resources usage and creation and name revelation. A Modal Logic, which is inspired by Hennessy-Milner Logic, is introduced for specifying and verifying properties of MLTS. To show usefulness of our proposal we show how MLTS can be used to describe the operational semantics of two formalisms with opposite objectives, namely Klaim and the asynchronous pi-calculus.

Sensoria integration and future plans.

For SOC, like for other formalisms, it is crucial to have tools for establishing deadlock freeness, liveness and correctness with respect to given specifications. However for programs involving different actors and authorities it is also important to establish other properties such as resource allocation, access to resources and information disclosure. For this purpose, the use of temporal logics like MoMo can be used for specifying and verifying dynamic properties of distributed services running over a wide area network.

In the months 13-30 we plan to use MoMo as the logic for reasoning about SENSORIA core calculi. We plan to develop prototype tools for supporting automatic verification of MoMo specifications, and we plan to extend MoMo for reasoning about systems whose implementation is not completely specified.

We want also define a methodology that permits refining specifications while preserving formulae satisfaction.

6 Type based Analysis of Service Reconfiguration

The overall framework

In current object-oriented programming practice, composition-based modularization seems to have become the most common structuring mechanism, reflecting a shift of programming style from a pure, inheritance-based object-oriented style, towards the so- called *component-based* or *service-based* programming idioms, which favor blackbox composition. Notwithstanding the proposal of many sophisticated type safe approaches to module and class composition, the mechanism most frequently used to structure object-oriented applications in the *component-oriented* style is the ad-hoc assembly of webs of objects (a.k.a. service providers), where individual elements refer to each other through references. In previous work, we had presented a programming calculus with the aim to capture essential ingredients of object-oriented component programming styles, such as explicit context dependence, subtype polymorphism at the level of both components and objects, late composition, and avoidance of inheritance in favor of composition. A type system was also defined, with types assigned to (first-class) components and objects, thus ensuring runtime safety of compositions. However, although in such a model components may be dynamically composed, the structure of objects gets fixed once for all at instantiation time, thus excluding any possibility of dynamic reconfiguration, which is a needed feature in high availability systems such as service based ones.

The work done

In this line of research we have developed a new core component-oriented programming language [2006-SCS], obtained by extending a lambda-calculus with imperative records with a minimal set of architectural primitives. Moreover, we develop a type system that statically enforces, besides the absence of usual runtime errors, consistency of component compositions and atomicity of dynamic reconfiguration.

Our design is semantically motivated by considering a domain of configurators, components, and objects; all such entities are first-class in our model. Intuitively, configurators correspond (by analogy) to the usual notion of *makefile*. Essentially, each configurator contains a series of instructions (architectural primitives) about how to assemble a component. It is expected that the soundness of any expressive notion of dynamic reconfiguration will turn out hard to ensure by purely static typing means, if one also wants to preserve object-level information hiding in the programming language. It is therefore important to explore the language design space involving combinations of static and dynamic checking, we believe to have isolated such an interesting combination. Thus, in our present proposal, type checking statically ensures good behavior of configurators, that is, that components built from well-typed configurators are architecturally consistent, and that objects instantiated from well-typed components are free from runtime errors.

Attached papers

[2006-SCS] João C. Seco and L. Caires. *Types for Dynamic Reconfiguration*. Proceedings of the European Symposium on Programming ESOP 06, Lecture Notes in Computer Science 3924, Springer-Verlag, 2006.
short abstract: We define a core language combining computational and architectural primitives, and study how static typing may be used to ensure safety properties of component composition and dynamic reconfiguration in object-based systems. We show how our language can model typed entities analogous of configuration scripts, makefiles, components, and component instances, where static typing combined with a dynamic type-directed test on the structure of objects can enforce consistency of compositions and atomicity of reconfiguration.

Sensoria integration and future plans.

The work on ”Type based analysis of Service Reconfiguration” will continue along the lines described above. We intend to extend the framework developed until now, which abstracts away from concurrency and distribution issues, and just focus on service composition, with concrete concurrency and distribution, and then with some form of behavioral types. While these issues may be tackled in the context of our component calculus, it would be probably much more interesting to investigate static analysis of reconfiguration in a service oriented calculus such as the one being proposed in WP2.

7 Approximating the control structure of processes

The overall framework

Recent years have seen an increased interest in applying static analysis techniques to highly concurrent languages, in particular a variety of process calculi allowing concurrent processes to interact by means of synchronization or communication. A common characteristic of many of these analyses is that they are mainly concerned with properties of the configurations of the models and to a lesser degree with properties of the transitions. In this work we focus on the transitions.

The work done

We study the classical approach of Data Flow Analysis where transfer functions associated with basic blocks are often specified as Bitvector Frameworks or, more generally, as Monotone Frameworks [2006-HRNFN] — what these analyses have in common is that there are ways of *removing* analysis information when no longer appropriate. We give the first account of an instance of an analysis problem for Milner’s CCS where suitable generalizations of the *gen* and *kill* components of Monotone Frameworks are used to define transfer functions that enable the construction of *finite automata* capturing the behaviour of processes — finiteness is achieved by incorporating ideas from yet another approach to static analysis, namely Abstract Interpretation.

To illustrate the development consider the following CCS process modelling a (unary) semaphore S:

$$S \triangleq g.\, p.\, S$$

First the process offers the action g, then the action p after which it starts all over again. Assume that it operates in parallel with a process Q given by

$$Q \triangleq \overline{g}.\, \tau.\, Q + \overline{p}.\, Q$$

that is willing to either perform the action $\overline{g}$ (that will synchronize with a g action) or the action $\overline{p}$ (that will synchronize with a p action). After the $\overline{g}$ action some internal action τ is performed and then the process recurses; after the $\overline{p}$ action the process recurses immediately. The semantics gives rise to the following infinite transition sequence:

$$S \mid Q \longrightarrow p.S \mid \tau.Q \longrightarrow p.S \mid Q \longrightarrow S \mid Q \longrightarrow \cdots$$

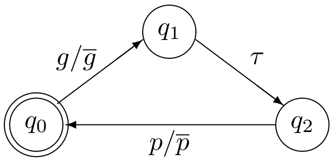
Each configuration is characterized by its *exposed actions*: initially g, $\overline{g}$ and $\overline{p}$ are ready to interact but only g and $\overline{g}$ actually do. In the resulting configuration $p.\, S \mid \tau.\, Q$ the actions p and τ are exposed and this time only τ can be executed, etc. Note that the action $\overline{p}$ is exposed in the first configuration but not in the second and p is exposed in the second but not in the first configuration. Thus once the actions participating in the interaction have been selected they will cause some actions to become exposed (e.g. p) while others no longer will be exposed (e.g. $\overline{p}$). In our analysis this will be reflected in the definition of transfer functions containing a *kill* component as well as a *generate* component much as in the classical Bit Vector Frameworks where the transfer functions take the simple form:

$$f_{\mathsf{block}}(E) = (E \setminus \mathit{kill}_{\mathsf{block}}) \cup \mathit{gen}_{\mathsf{block}}$$

Typically E is a *set* containing the data of interest as for example reaching definitions: which variable definitions may reach a given program point.

The general situation is more complex as there might be several occurrences of the same action that are exposed — indeed there might be an infinite number of such actions and as a consequence we shall use *extended multisets*. In the classical Bit Vector Frameworks the generated and killed information is always precise; in the present setting we shall need to approximate it. For the generated actions we shall use an *over*-approximation as it is always safe to say that more actions are exposed. However, for the killed actions we need an *under*-approximation: it is always safe to remove fewer actions than necessary.

Based on this information the analysis constructs a finite automaton representing the control structure of the process. Each state in the automaton corresponds to an extended multiset of exposed actions for a configuration and the transitions of the automaton reflect the possible interactions between these actions. For the process $S \mid Q$ above we obtain the automaton



where the initial state q_0 corresponds to g, $\overline{g}$ and $\overline{p}$ being exposed, q_1 to p and τ being exposed and q_2 to p, $\overline{g}$ and $\overline{p}$ being exposed. The annotations on the edges then reflect the interactions being performed.

The analysis uses a simple *worklist algorithm* to construct the automata. The initial state corresponds to the exposed actions of the initial process and more and more states are added as the need arises. It is crucial to reuse states whenever possible, that is, when they represent the same exposed actions as otherwise the construction need not terminate; in order to handle this we borrow the *widening* technique from Abstract Interpretation.

Attached papers.

[2006-HRNFN] Hanne Riis Nielson and Flemming Nielson: *Data flow analysis for CCS.* Submitted for Festschrift for Reinhard Wilhelm.
short abstract: This paper gives an overview of the approach to analyzing the control structure of CCS processes.

Sensoria integration and Future plans.

The work is very central for the WP3 task on behaviours in that the analysis indeed extracts information about the behaviour of processes. The work presented has been developed for CCS and it is planned to extend it to more expressive calculi considered in the SENSORIA project and to apply it to processes for services.

8 Graph Transformation Systems

The overall framework

Graph transformation systems are a flexible formalism for the specification of the operational behaviour of complex systems, that may take into account aspects such as object-orientation, concurrency, mobility and distribution. In fact, graphs can be naturally used to provide a structured representation of the states of a system, which highlights its subcomponents and their logical or physical interconnections. Then, the events occurring in the system, which are responsible for the evolution from one state into another, are modelled as the application of suitable transformation rules. Such a representation is precise enough to allow the formal analysis of the system under scrutiny, as well as amenable of an intuitive, visual representation, which can be easily understood also by a non-expert audience.

There are several approaches to graph transformation proposed in the literature, which differ in the way the rules and their application to graphs are defined. Even if the simplest way to define graph transformation is by resorting to standard set-theoretical notions, a more abstract, algebraic description based on simple categorical constructions is often preferred, as it allows to reason about graphs and their transformation abstracting out from the concrete identity of nodes and edges, and it makes easier the proof of relevant properties.

The work done

There are two main categorical approaches used to describe the effect of applying a rule to a graph. These are the double-pushout approach (DPO) and the single-pushout approach (SPO). Both approaches use concepts of category theory to obtain an elegant and compact description of graph rewriting, but they differ with respect to the kind of morphisms under consideration, the form of the rules, and the diagrams the rewriting steps are based on.

In [2006-CHHK] we have proposed a new categorical approach to rewriting that combines the good properties of both approaches and improves them by allowing to model cloning of structures in a natural way. The new approach is called “sesqui-pushout (SqPO) rewriting”, because it can be placed, conceptually, halfway between the single-pushout and the double-pushout approach (*sesqui* means *one-and-a-half* in Latin). Differently from the DPO approach, SqPO rewriting is deterministic even if the rule to be applied is not left-linear, and it is able to model *deletion in unknown context*, a typical feature of the SPO approach. Besides the basic definitions of the new approach, its precise relationship with the existing approaches has been investigated, and some first results about parallelism and local confluence have been worked out.

In the classical theory of Petri nets, Condition/Event (C/E) Systems are safe nets where the enabling condition for a transition requires, additionally, that no token is present in the post-set. Thus, at any time a subset of all conditions are true (those holding a token), and firing a transition modifies this subset. In [2006-CHS] we are developing a theory of *Subobject Transformation System*. These can be seen as a generalization of C/E Systems, where instead of having a *sub-set* of the “conditions” we have a *sub-object* of conditions for a fixed object T in an arbitrary adhesive category. A subobject of T can be transformed into another one by applying a rule of the system. This theory is a distilled, more restricted version of the Double-Pushout Approach in an adhesive category, which has nevertheless several concrete instances already studied in the literature.

Reflexive systems arise naturally in many areas where graph transformation techniques have been applied with success, like biological and chemical systems, distributed and mobile computing, service oriented architectures, and data type specifications. Moreover, the reflexive extension of many different kinds of rewrite systems have been considered in the literature. In particular, dynamic nets have been proposed as a mobile extension of Petri nets, expressive enough to model mobile calculi like pi-calculus and join calculus. Dynamic nets are indeed strictly more expressive than Petri nets. Exploiting the analogy between Petri nets and Graph Grammars, we propose a reflexive extension of Graph Grammars (under the double-pushout approach), called Dynamic Graph Grammars (DynGGs) [2006-BM], whose productions can release fresh parts of the type graph and new rewrite rules. However, when posing the question:

"Are Dynamic Graph Grammars more expressive than ordinary ones?"

our main result provides a negative answer: though DynGGs can offer a more convenient abstraction, computationally speaking they are not more expressive than ordinary Graph Grammars.

Attached papers.

[2006-CHHK] A. Corradini, T. Heindel, F. Hermann, B. Koenig. *Sesqui-pushout rewriting*. In Proc. Third International Conference on Graph Transformation (ICGT'06), LNCS, Springer, 2006. To appear.
Short abstract: *Sesqui-pushout (SqPO) rewriting* is a new algebraic approach to abstract rewriting in any category.
SqPO rewriting is a deterministic and conservative extension of double-pushout (DPO) rewriting, which allows to model "deletion in unknown context", a typical feature of single-pushout (SPO) rewriting, as well as cloning.
After illustrating the expressiveness of the proposed approach through a case study modelling an access control system, we discuss sufficient conditions for the existence of final pullback complements and we analyze the relationship between SqPO and the classical DPO and SPO approaches.

[2006-BM] R. Bruni, H. Melgratti. *Dynamic Graph Transformation Systems*. In Proceedings of ICGT 2006, 3rd International Conference on Graph Transformation, LNCS, Springer, 2006. To Appear.
Short abstract: We introduce an extension of Graph Grammars (GGs), called *Dynamic Graph Grammars* (DynGGs), where the right-hand side of a production can spawn fresh parts of the type graph and fresh productions operating on it. The features of DynGGs make them suitable for the straightforward modeling of reflexive mobile systems like dynamic nets and the join calculus. Our main result shows that each DynGG can be modeled as a (finite) GG, so that the dynamically generated structure can be typed statically, still preserving exactly all derivations.

Sensoria integration and Future plans.

Graph Transformation Systems are used, as specification and modeling formalism, in several work packages of SENSORIA (for example, T1.3: Service-Oriented Business Modelling, T3.3: Behavioural Properties, T5.1: Primitives for Service Level Agreement, T5.3: Dynamic Reconfiguration, and T7.1: Transformation and Refinement Methods) The work described above is mainly concerned with foundational aspects of graph transformation systems, improving the expressiveness of the formalism. As such, it will potentially influence the use of GTS in other WPs of the project.

The work on the foundation of GTS will continue along the topics described above. The greater expressive power of the new SqPO approach will be experimented by modeling a subset of the Core Calculus.

9 Conclusion and further plans

The activity done so far actually spawns in several directions and covers a wide range of techniques. This is quite natural for this first period of the project where the partners activities are more oriented to the analysis of the fitness of the various techniques in which they have greater expertise with respect to the class of problems in which SENSORIA is interested. For the next SENSORIA period (months 13-30), as one or more common SENSORIA languages (Core Calculus) will become stably defined it is expected they will provide an attractive playground for the investigated techniques allowing to achieve a greater and common focus with respect to SENSORIA goals for the qualitative evaluation of systems.

Building on formalisms and techniques studied during the first year of the project, during months 13 to 30 we will therefore continue to develop logics and other formalisms for description and analysis of behavioural properties of services. In particular, we will study temporal logics for expressing behavioural properties of service oriented systems also as modelled in various service-oriented calculi/languages. We will apply process calculi to the study of behavioural properties of web service composition and compensations. Moreover, we will study some basic questions concerning the discriminating power of the Spatial Logic, focusing on the equivalence on processes induced by the logic. We will also present some related (un)decidability results. We plan to extend the MoMo logic for reasoning about systems whose implementation is not completely specified. We want also to define a methodology that permits refining specifications while preserving formulae satisfaction. Finally, we will develop a generic logical system for specification of component systems, formalise it as an institution and apply it to description of behavioural properties of services.

10 Relevant Sensoria Publications and Reports

GMaica S.Gnesi and F. Mazzanti *A Model Checking Verification Environment for UML Statecharts* Proceedings XLIII AICA Annual Conference, University of Udine - October 2005, AICA.

SLMC Hugo Vieira, Luís Caires and Ruben Viegas; *The Spatial Logic Model Checker User's Manual* v1.0, Technical Report of Departamento de Informatica, FCT/UNL. November 2005.

BGMM M.H. ter Beek and S. Gnesi and F. Mazzanti and C. Moiso, *Formal Modelling and Verification of an Asynchronous Extension of SOAP*. ISTI Technical Report ISTI-2006-TR-19, June 2006

BM R. Bruni, H. Melgratti. *Dynamic Graph Transformation Systems*. In Proceedings of ICGT 2006, 3rd International Conference on Graph Transformation, LNCS, Springer, 2006. To Appear.

CHHK A. Corradini, T. Heindel, F. Hermann, B. Koenig. *Sesqui-pushout rewriting*. In Proc. Third International Conference on Graph Transformation (ICGT'06), LNCS, Springer, 2006. To appear.

DEQ Luís Caires and Hugo T. Vieira. *Extensionality of Spatial Observations in Distributed Systems*, proc. 13th International Workshop on Expressiveness in Concurrency (EXPRESS'069 June 2006.

DNL R. De Nicola and M. Loreti, *Multi Labelled Transition Systems for Nominal Calculi and their Logics* (Submitted).

HRNFN Hanne Riis Nielson and Flemming Nielson: *Data flow analysis for CCS.* Submitted for Festschrift for Reinhard Wilhelm.

MUG33 F.Mazzanti *UMC 3.3 User Guide* ISTI Technical Report 2006-TR-33, September 2006.

SCS João C. Seco and L. Caires. *Types for Dynamic Reconfiguration*. Proc. of the European Symposium on Programming ESOP 06, LNCS 3924, Springer-Verlag, 2006.

Zawa A. Zawłocki, *Diagram Models for Interacting Components*, Submitted. Extended abstract presented on 3rd International Workshop on Formal Aspects of Component Software, FACS'06, Prague, Czech Republic, September 20-22, 2006.

Zawb A. Zawłocki, *An Institution for Interacting Components*. Submitted. Extended abstract presented on 18th International Workshop on Algebraic Development Techniques, WADT'06, La Roche en Ardenne, Belgium, 1-3 June 2006.

References

[1980-HM] M.Hennessy and R.Milner *On observing nondeterminism and concurrency*. Procs ICALP 80 LNCS 85 295-309

[1980-PTL] D.Gabbay, A.Pnueli, S.Shelah and J.Stavi *On the temporal analisys of fairness* Proc. ACM Symp. on Principles of programming Languages (1980) 163-173

[1981-CE] E.M. Clarke and E.A.Emerson *Design and synthesis of synchronization skeletons using branching time temporal logic* LNCS 131 pp 52-71

[1983-K] Kozen,D. *Results on the Propositional mu-calculus* Theoretical Computer Science 27 (North Holland 1983)

[1986-EH] CTLstar-E.A.Emerson and J.Halpern *Sometimes and not never revisited: On branching versus linear time temporal logic* Journal of the ACM (JACM), v.33 n.1, p.151-178, Jan. 1986

[1990-DNV] R. De Nicola and F.W. Vaandrager, *Action versus state based logics for transition systems*. Proceedings of the LITP spring school on theoretical computer science on Semantics of systems of concurrent processes LNCS 469 1990.

[2005-DL04] Rocco De Nicola and Michele Loreti *MoMo: A Modal Logic for Reasoning About Mobility*. Proc. of FMCO 2004, Revised Lectures, pp95-119, LNCS 3657, Springer, 2005.

[2005-FMCO] Luís Caires *Composing Distributed Services with Spatial Types*. Invited Talk at the Formal Methods for Components and Objects (FMCO 2005), Amsterdam, November 2005.

[2005-SLMC] Hugo Vieira, Luís Caires and Ruben Viegas; *The Spatial Logic Model Checker User's Manual* v1.0, Technical Report of Departamento de Informatica, FCT/UNL. November 2005

[2005-GMaica] S.Gnesi and F. Mazzanti *A Model Checking Verification Environment for UML Statecharts* Proceedings XLIII AICA Annual Conference, University of Udine - October 2005, AICA.

[2006-CHHK] A. Corradini, T. Heindel, F. Hermann, B. Koenig. *Sesqui-pushout rewriting*. In Proc. Third International Conference on Graph Transformation (ICGT'06), LNCS, Springer, 2006. To appear.

[2006-CHS] A. Corradini, F. Hermann, P. Sobocinski. *Subobject Transformation Systems*. Draft.

[2006-BM] R. Bruni, H. Melgratti. *Dynamic Graph Transformation Systems*. In Proceedings of ICGT 2006, 3rd International Conference on Graph Transformation, LNCS, Springer, 2006. To Appear.

[2006-DEQ] Luís Caires and Hugo T. Vieira. *Extensionality of Spatial Observations in Distributed Systems*, proc. 13th International Workshop on Expressiveness in Concurrency (EXPRESS'069 June 2006.

[2006-SCS] João C. Seco and L. Caires. *Types for Dynamic Reconfiguration*. Proceedings of the European Symposium on Programming ESOP 06, LNCS 3924, Springer-Verlag, 2006.

[2006-Zawa] A. Zawłocki, *Diagram Models for Interacting Components*, Submitted. Extended abstract presented on 3rd International Workshop on Formal Aspects of Component Software, FACS'06, Prague, Czech Republic, September 20-22, 2006.

[2006-Zawb] A. Zawłocki, *An Institution for Interacting Components*. Submitted. Extended abstract presented on 18th International Workshop on Algebraic Development Techniques, WADT'06, La Roche en Ardenne, Belgium, 1-3 June 2006.

[2006-BGMM] M.H. ter Beek and S. Gnesi and F. Mazzanti and C. Moiso, *Formal Modelling and Verification of an Asynchronous Extension of SOAP*. ISTI Technical Report ISTI-2006-TR-19, June 2006

[2006-MUG33] F.Mazzanti *UMC 3.3 User Guide* ISTI Technical Report 2006-TR-33, September 2006.

[2006 -DNL] R. De Nicola and M. Loreti, *Multi Labelled Transition Systems for Nominal Calculi and their Logics* (Submitted).

[2006-HRNFN] Hanne Riis Nielson and Flemming Nielson: *Data flow analysis for CCS.* Submitted for Festschrift for Reinhard Wilhelm.