NSF-CNPq Collaborative Research - NSF 98-139

Mathematical and Engineering Foundations

for

Interoperability via Architecture

U.S. Partners

John McCarthy/Carolyn Talcott (Stanford University, Stanford, California) and José Meseguer (SRI, Menlo Park, California)

Brazilian Partner

Armando Haeberer (Pontifical Catholic University of Rio de Janeiro, Rio de Janerio, Brazil)

Previous Interactions

Discussions of mutual interests and existing formalisms at workshops in Rio de Janeiro (May 1997) and New Orleans (November 1997). Visit of Professors Tom Maibaum and José Fiadeiro to SRI and Stanford summer of 1998 (as associates/members of the Laboratory of Formal Methods).

Bio and indication of submission to CNPq from Brazilian Partner See attached.

A. Project Summary

NSF-CNPq Collaborative Research

Mathematical and Engineering Foundations for Interoperability via Architecture

Large software systems are increasingly complex and costly. Furthermore, their maintenance and evolution account for a large percentage of the overall cost. The design, development, and maintenance of a system can be greatly helped by good documentation of its *architecture*, that is, of how it is structured into meaningful subsystems or *components*, and how those subsystems are glued together to form the overall system.

However, to take full advantage of the architectural structure it is important to have a wide range of rigorous semantics-based descriptions of diverse aspects or views of the system that can be associated to components of an architecture. This includes, for example, both executable and non executable formal specifications, models of concurrency and interaction, abstractions suitable for system analysis and model-checking, and specification of security, dependability, and real-time requirements. In this way, architectural designs can be subjected to different kinds of analysis, validation, and prototyping techniques to identify errors and inconsistencies as early as possible, before substantial implementation efforts are wasted on realizing the wrong design.

A key unresolved problem is the interoperability problem, namely, how the architectural pieces and their multiple and possibly heterogeneous semantic descriptions corresponding to the different views fit together to make a coherent whole, and how these different descriptions impose constraints on each other.

The main objectives of this collaborative project are (1) developing mathematical foundations for integrating and interoperating in a coherent and mathematically rigorous way diverse architectural descriptions, formal specifications, and executable prototypes of object-based distributed systems and (2) based on such foundations, developing a methodology for system design, development, and evolution that supports a seamless integration of the different formal and informal system descriptions and makes explicit the constraints that these different views of a system impose on each other.

The development of mathematical foundations will begin with an in-depth study of a wide range of notations and formalisms and of their mutual relationships that seem particularly promising for describing systems at different levels. The formalisms chosen to populate and explore the formal architectural space will include semantic models of concurrency and interaction, high-level specification logics, engineering design notations, object calculi, and executable specification languages. Rewriting logic will play an important role, not only as an executable specification formalism, but also as a reflective metalogical framework in which the chosen formalisms and their relations can be formalized. These formalizations can be executed using the meta tools developed in the Maude language, an implementation of rewriting logic.

The development of a methodology based on these foundations will be driven by substantial case studies in which the ensemble of formalisms and their systematic relationships will be applied to and tested on concrete applications that demonstrate the interoperation of the formalisms and the subsequent benefits.

It is expected that these foundations and experimental methodology will advance the state of the art in software development and evolution, and will lead to new tools and methods that, when used properly, will substantially reduce the cost and effort of software development and maintenance.

C. Project description

1 Introduction

To manage the complexity of large, dynamically evolving software systems it is essential to structure them, preferably before the fact, into meaningful, conceptually manageable parts and to have a clear understanding of the role of each part and how the parts combine to make the whole. Decomposition happens in many dimensions, not necessarily orthogonal or with disjoint parts. The same components may play several different roles in a single system. Different views of components may overlap.

There is a growing number of notations and formalisms for describing different aspects of software systems. Some are graphical, some are executable, some are syntactic while others are semantic, some express simple properties that can be machine checked, some express complex high-level properties that can not in general be automatically checked. In current OOD practice graphical notations are mainly used to design objects and describe their behavior. In the emerging field of software architecture, similar graphical notations are used to describe entire system designs.

Architectural descriptions provide a way of understanding how a complex system is built and of managing its complexity. Some existing formalisms provide for modular descriptions, and some specification frameworks have notions of both refinement and composition. However, a real system is likely to have descriptions of different parts using different formalisms based perhaps on different models of computation and interaction. There are also likely to be descriptions of different aspects (ilities) of the same component using different notations or formalisms. What is lacking is a rigorous mathematical foundation for understanding mappings and relations among the various formalisms at an architectural level. We need to be able to combine and mix formalisms in semantically meaningful ways and to preserve and reflect the architectural structure of a system.

A semantic basis for composition and interoperation should permit determining what parts are interchangeable under given conditions, and to predict effects of changes: of components on the whole and of system requirements on requirements for the parts. To achieve sensible, correct, and secure interaction between components, there are needs for

(i) a better semantic understanding of interoperation frameworks

(ii) a better semantic understanding of interconnection, coordination, and architecture description languages

(iii) a semantic foundation for fitting together the various languages and interoperation models

(iv) a richer semantic basis for interoperation of formal systems themselves – needed, for example, in

   (a) formalizing different aspects of complex systems and reasoning across such formalizations
   (b) correct interoperation of formal tools such as theorem provers, declarative languages, and analyzers

There is a wide range of applications where such a semantic foundation for interoperation can be of value. It is especially important in distributed applications that combine components from diverse domains such as data bases, spreadsheets, knowledge bases, multimedia, mobile agents, control theory, or event-based simulations.

Figure 1: The Big Picture.

The multiple dimensions of the space of system descriptions and their relationships are illustrated in figure 1. The picture shows two planes, one linguistic and the other semantic, as well as rewriting logic as a representational framework. In the linguistic plane we see notations being used to describe both components and the architectural structure of the system. The notations can be informal or formal. Such notations may include object-oriented design notations, architectural description languages, coordination languages, formal specification languages (executable or not), and programming languages. These possibly heterogeneous descriptions can be related vertically by refinement relations, and horizontally by architectural composition operations. In the semantic plane we see a variety of operational models of the system and its components. Such models may include transition systems, event-based models, categorical models, and net-based models, among others. These models are related by refinements and architectural compositions mirroring those in the linguistic plane. The correspondence between the two planes is one of denotation that respects the relations in an appropriate sense. The role of rewriting logic as a representational framework is to represent formally the different notations and the mappings between them in a way that supports heterogeneous interoperation. The model theory of rewriting logic allows models in the semantic plane to also be formally represented.

The proposed research has two main objectives, one foundational and the other methodological:

(1) Make the picture in Figure 1 mathematically precise, yielding a basic framework for heterogeneous formal interoperation via architecture. The resulting mathematical framework will formalize the objects and arrows in the picture and will provide a detailed rigorous understanding of their relationships.

(2) Instantiate the picture with specific models, formalisms, notations, and languages and carry out substantial case studies to illustrate the use and the benefits of such a mathematical foundation. The main fruit to be obtained out of such case studies is a new *methodology* rigorously based on the new mathematical foundations that will allow a seamless interoperation of different system descriptions and views and will make explicit their inter-relationships.

The specific choice of formalisms to instantiate the framework is to be determined. The collaborators bring expertise in a wide range of formalisms and frameworks and are actively advancing the state of the art in many directions. An overview of the relevant work and techniques can be found in § 2. The instantiating formalisms and case studies will be carefully chosen to address issues relevant to modern software development such as heterogeneity (in platforms, languages, and models), concurrency and distribution, mobility, openness and the need for adaptability and evolution.

2 Technical Background

This section provides an overview of some existing formalisms and frameworks and their relations that will serve as a starting point for understanding and populating the mathematical space that is the subject of the proposed research. The overview focuses on work of the members of the proposed collaboration and closely related work.

2.1 ARTS

The ARTS project at PUC-Rio [45, 42] is an ongoing attempt to bring together software theory and practice. ARTS' goal is the definition and implementation of a set of languages, models, theories, and tools for the method-independent development of real-time object-oriented systems.

The basic architecture of ARTS consists of two planes. In the user plane, software designers use engineering notations to model applications and libraries, via annotated class relationship, object communication, state behavior, and scenario diagrams. In the formal plane, normally unseen to users, steps taken during design are verified, guaranteeing design consistency. Underlying platforms provide mappings between the engineering notations and formal theories [12]. Designs are automatically coded in an executable language (currently DDL [11], a very high-level textual design description language representing the object-oriented discourse adopted, and then automatically transformed into C++ code).

Also essential to the architecture of ARTS is the ability to plug-and-play components, as for example diagram editors, target languages, and transformation systems. In this way ARTS can serve as a testbed for experiments of various natures, including experiments that explore the role of architectural description languages. Such languages could be considered as extensions to DDL, and could incorporate design and architectural patterns. Experimentation with frameworks is also of interest, regarding development, description, and presentation for instantiation.

2.2 Fork Algebras

Fork Algebras are a particular kind of Quasi-Projective Algebras. Using Fork Algebras a finitely axiomatizable algebra of First-Order Logic with Equality was constructed [44], as well as algebras of modal logics and of a modal action logic extended with time intervals which is the basis of an *object calculus* [5]. Programming calculi were constructed using Fork Algebras, and taking advantage of their *reflexiveness*, design strategies were defined and used for guiding program calculation at the same linguistic level as the calculation itself. Program calculation and strategy derivation (and use) were combined in the calculation of *general algorithms* solving classes of problems from were a specific algorithm solving a particular problem of this class can be easily derived. Specifying and calculating with Fork Algebras offers the advantage of the equational reasoning, which for a wide class of problems and approaches (as the rewriting one) is very convenient.

2.3 Rewriting Logic

Rewriting logic was first proposed by Meseguer as a unifying framework for concurrency in 1990 [69, 62]. Since then a large body of work, including more than 100 research papers and three language implementations, by researchers around the world has contributed to the development of several aspects of the logic and its applications in different areas of computer science. The recent proceedings of the First and Second International Workshops on Rewriting Logic and its Applications that have taken place in Asilomar, California [63] and Pont-à-Mousson, France [47] provide a good overview of current research directions.

Rewriting logic expresses an essential equivalence between logic and computation. System *states* are in bijective correspondence with *formulas*, and concurrent *computations* are in bijective correspondence with *proofs*. Given this equivalence between computation and logic, a rewriting logic axiom of the form $t \longrightarrow t'$ has two readings. Computationally, it means that a fragment of a system's state that is an instance of the pattern t can *change* to the corresponding instance of t' concurrently with any other state changes; logically, it just means that we can derive the formula t' from the formula t. Rewriting logic is entirely neutral about the structure and properties of the formulas/states t. They are entirely *user-definable* as an algebraic data type satisfying certain equational axioms. Because of this ecumenical neutrality, rewriting logic has, from a logical viewpoint, good properties as a *logical framework*, in which many other logics can be naturally represented. And, computationally, it has also good properties as a *semantic framework*, in which many different system styles and models of concurrent computation and many different languages can be naturally expressed without any distorting encodings.

As a *semantic framework*, rewriting logic has been shown to express very naturally a wide range of models of concurrency including CCS, the π-calculus, Petri nets, Actors, the UNITY language, parallel functional programs, dataflow, parallel graph rewriting, and concurrent object systems [62, 64, 73]. It has also been show to be a very flexible framework to specify both programming languages and architectural description languages (ADLs) [73] (see §2.7 for a more detailed discussion of ADL semantics).

As a *reflective logical framework* rewriting logic also enjoys very good properties [58, 73]. Being reflective, it can represent at the object level both its own metatheory and that of other logics and languages [17]. In this way, rewriting logic can be used as a *metalanguage* in which many different logics and languages can be specified and executed. The logical and semantic framework capabilities of rewriting logic will be systematically exploited in the proposed project to relate and interoperate the different notations and formalisms used to describe a system at different levels. In this regard, besides reflection itself, a crucial feature of rewriting logic is its *executable* nature, so that rewriting logic specifications become executable prototypes of the languages, logics, and mappings between their representations in the framework. This is well-supported by the Maude language [70, 15].

2.4 Actor Models for Open Distributed Systems

Actors are a natural basis for modeling open distributed systems. They provide units of encapsulation of control and state, interaction via asynchronous message passing, and dynamic architecture. Actor theories [86, 85, 60] were developed by Talcott as a mathematical framework for specifying actor systems at many levels of detail. In [85] a rewriting logic semantics for actor systems that are axiomatized by actor theories is given. Actor systems are extended by an algebra of *components*, where a component is formalized as a distributed subsystem that can include actors, messages, and other (sub-)components and that is encapsulated by a precisely defined interface that mediates its interactions with an open environment. The standard rewriting logic semantics is extended to include infinite behaviors in order to treat fairness.

Two other works proposing the use of rewriting logic as a semantic foundation for open distributed computation are Najm and Stefani [79] and Nakajima [80]. Najm and Stefani [79] have used rewriting logic to give a formal semantics to the operational subset of the Reference Model for Open Distributed Processing (RM-ODP) [46]. RM-ODP specifies a language-independent computational model underlying many of the distributed computational models that have been recently proposed, including those adopted for CORBA, the Network Object system, and the Java RMI system. Nakajima [80] shows how the calculus of mobile ambients of Cardelli and Gordon [10] can be naturally expressed in rewriting logic, and then uses rewriting logic specifications in CafeOBJ to formally specify the architecture of WEB-NMS, a Java/ORB implementation of a network management system with four architectural components.

Several semantic models of actor systems viewed as components of open distributed systems have been developed by Talcott [86, 87, 88]. These models express the semantics of components in terms of their interactions with the environment without any mention of a component's global state. Possible interaction scenarios are represented as sets of interaction events with partial or total orderings. To support modularity in specification, reasoning, and coding, the notions of *actor component algebra* and *actor component morphism* were introduced. Components model actor systems at different levels of abstraction, ranging from simple interfaces to syntactic descriptions of system configurations to sets of behaviors in a semantic domain. Morphisms from syntactic to semantic algebras give a wide range of compositional semantics. In particular, semantics of configurations of an actor theory are given by such morphisms. Other morphisms are useful for transferring results between algebras.

Specification diagrams [83] are a graphical notation for specifying the behavior of actor system components. They are similar in spirit to OMT/UML notations such as event sequence diagrams [40, 81]. Specification diagrams are given a formal semantics by interpreting the notation as presenting an actor theory. Methods for proving satisfaction and equivalence relations between specifications and other descriptions of actor systems (program modules, other actor theories) by actor theory transformations are being developed. Preliminary results are reported in [84]. Additional work is in progress to develop a richer algebra of actor theories to support this methodology.

2.5 Specification of Object Systems

Rewriting logic describes concurrent systems in terms of their states and rules specifying how the system may evolve. Here deduction is synonymous with computation. Another possible use of logic in the context of concurrent systems is to talk about change by allowing properties of the behavior of systems to be stated without having to define how they should be realized. Typical logics that have been used for this purpose are modal logics, in particular temporal logics (e.g. [53]). The role of deduction in this case is to support the derivation of new properties from a kernel presentation that constitutes the specification of the desired system. That is to say, a system (or system component) is still viewed, from a logical point of view, as a theory (presentation) but one in which the axioms do not state directly what changes are performed on the system state but, instead, which properties these changes are required to satisfy.

2.5.1 Rewriting Logic Specification of Object Systems

Concurrent object-based systems can be very naturally specified and executed in rewriting logic [70]. The case of *asynchronous* object systems such as actors has already been discussed in §2.4, but *synchronous* object systems or systems combining synchrony and asynchrony can be specified with equal ease. Class inheritance has also a precise semantics in terms of the corresponding rewriting logic specifications [70]. This style of executable object-oriented specification is well-supported by the Maude language that provides special syntax for such specifications through its object-oriented modules [70].

M. Wirsing and A. Knapp [92] have argued lucidly about the practical importance of combining object-oriented design notation with formal executable specifications. Their proposal is to annotate diagrams in an existing and widely used object-oriented notation and then derive formal executable rewriting logic specifications written in an extension of Maude. The specifications can then be used to execute and formally analyze the designs, and even to generate code in a conventional language such as Java. The importance of combining object-oriented design notations with formal specifications has also been acknowledged in the most mainstream modeling language, namely UML, that has an explicit place holder for such specifications as annotations.

2.5.2 Modal/Temporal Logic Specification of Object Systems

To support the development of reactive systems, specifically the role of modularity, it is not enough that the specification logic has the expressive power required to express the desired properties. It is also necessary to support the specification of individual components and of their interconnections as a means of building the specification of a large system. The work done by Fiadeiro, Maibaum and their collaborators [37, 34, 36] has focused precisely on this issue. More precisely, they have investigated the use of categorical techniques, known from General Systems Theory and Abstract Data Type Specification, in combination with modal logics for formalizing the process of building systems from interconnected components. This combination of logical and categorical techniques has also been applied to parallel program design languages in the style of UNITY [14] and IP [41], providing semantics for modularization techniques based on the notion of superposition. This has resulted in the development of a programming design language called *Community* [33].

Two formalisms that provide explicit support for object systems and can reason about their rewriting logic specifications have been recently developed. One is a version of the modal μ-calculus proposed by Lechner [48, 49] for reasoning about object-oriented Maude specifications. Another is Denker's object-oriented distributed temporal logic DTL^+ [24, 22], that extends the DTL and D_1 distributed object temporal logics of Ehrich and Denker [30, 23, 29].

Lechner [48, 49] uses her version of the modal μ-calculus to identify several safety properties of Maude object-oriented specifications that can be characterized as special classes of formulas. She has also done a fine analysis of how certain reuse properties such as inheritance, subconfiguration, and message combination can be reflected at the modal μ-calculus level by the preservation of certain classes of formulas. Furthermore, she has verified properties of useful examples using her logic, and has shown how modal μ-calculus specifications can be *refined* into more specific ones until reaching a rewriting logic specification.

Denker's DTL^+ [24, 22] builds upon her previous work on temporal logic specification of object-oriented distributed information systems and has the following aims: (1) DTL^+ offers several communication concepts, including synchronous and asynchronous communication, and (2) DTL^+ is designed in such a way that the principles of object-oriented specifications based on rewriting logic are captured. In particular, DTL^+ can be used to reason about properties of object-oriented Maude specifications. In DTL^+ the underlying rewriting logic is extended by temporal operators.

Duarte [27] has developed a category of actor specifications based on a branching time temporal logic in which individual actor behaviors can be specified, component specifications can be composed to form specifications of larger systems, and higher-level properties of specified actors systems can be stated and formally proved. The theoretical foundation for this work builds on ideas for object specification of [37]. A rely-guarantee discipline and proof principles are proposed for modular reasoning about systems. As a nontrivial case study, Duarte has applied the formalism and associated design discipline to the design of a location management architecture for networks of mobile users and devices and verification of system properties [26]. The use of this actor specification category as a framework for refining high-level specifications into lower-level (executable) specifications seems quite promising. Intuitively there is an embedding relation between actor theories and Duarte's logic analogous to that between object-oriented specifications in rewriting logic and Denker's DTL^{+}.

2.6 Graph- and Net-based Models for Concurrency

Like rewriting logic theories, graph grammars are a powerful and expressive means of specifying complex concurrent and reactive systems, supporting a wide range of concurrency models and providing specification features like compositional parallel composition operators and refinement techniques. Using graph grammars, both concurrency and persistence of identities are naturally represented via graph grammars. Therefore, graph grammars seem to be a promising formalism for the description of concurrent and reactive systems. In particular, object-systems like Actor systems can be described using graph grammars. A potentially fruitful mapping between logics would be an embedding of graph grammars in rewriting logic, providing rewriting logic with subtheories that account for persistence while naturally enriching the class of graph structures available to the graph grammar formalism. The basis for this has been initiated in [64].

Two important issues for semantic modeling of concurrent objects and object interaction are reification and encapsulation. Correct understanding of encapsulation requires some form of internal nondeterminism, and an adequate treatment of reification should be centered around the notion of transaction (instead of action). Moreover, for giving semantics for some object-oriented languages, diagonal compositionality is an essential property, that is, reifications compose (vertically) and distribute over combinators (horizontally). Petri nets, labeled transition systems, asynchronous transition systems, and other well-known net-based systems do not satisfy diagonal compositionality. Nonsequential automata (first introduced in [61]) constitute a categorical semantic domain based on labeled transition systems with full concurrency, where restriction and relabeling are functorial and a class of morphisms stands for reification. It is a model for concurrency which satisfies the diagonal compositionality requirement. Categories of Petri nets and nonsequential automata are unified through adjunctions extending the approach in [82]. A challenging problem is the clarification of the relationship between logics and net-based models and its compositional aspects.

2.7 Software Architecture

Two important questions left unanswered in the present state of the software architecture are (1) the formal *semantics* underlying a particular ADL, and (2) how to *interoperate* different ADLs. The importance of having a formal semantics is not, as it might appear, an academic exercise; it is eminently practical. Without such a semantics an architectural notation becomes badly crippled, because it becomes impossible to subject the designs it expresses to any meaningful *formal analysis*; that is, any application of formal methods is out of the question. One way of understanding why this is bad in practice is to realize that *if* one had a formal semantics for a design in the form of an *executable specification* one could in fact execute such a design, finding many bugs and inconsistencies early in the design process. The need for interoperation is related to the fact that different subsystems may have widely different architectural structures, and therefore they may be best expressed in different architectural styles and languages. Of course, the need for a formal semantics reappears in an even more intense form, because now it is not even enough to have a semantics for each of the ADLs. It becomes imperative to systematically relate those semantic accounts in order to achieve a semantic form of interoperation supporting, for example, the execution of the whole design by a combined execution of executable specifications for each of the subsystems.

Meseguer and Talcott have proposed rewriting logic as a promising semantic framework for ADLs and their interoperation [65, 66] that provides an answer to the above two questions about how to give ADLs a formal semantics and how to interoperate them. Typically, each architectural style has a different underlying model of computation and interaction as its semantic basis. Such models can naturally be represented as rewrite theories of a particular form, and designs of different styles can then be mapped to theories in the appropriate style. The resulting theories can be easily combined and executed in a resulting overall rewrite theory. These ideas have been illustrated by means of a nontrivial case study involving several architectural styles such as pipes-and-filters, dynamic architectures, implicit invocation, and message passing, and the corresponding rewriting logic specifications have been executed and interoperated in Maude.

The categorical semantics of parallel program design discussed in §2.5.2 was also used in [36, 32, 35, 38, 39] for formalizing the notion of architectural connector in the sense of [4]: the roles and the glue of a connector are objects (programs) in the relevant category, and the morphisms provide the connections between the roles and the glue. The instantiation of the roles with given components is achieved via a mechanism similar to parameterization as used in Abstract Data Types. This work has also shown how the notion of connector can be generalized by choosing the roles to be defined in a specification logic and the roles and glue in a program design language.

2.8 Relations and Synergies

The formalisms and technologies discussed earlier in this section provide a broad coverage of the different forms and levels of formal description of software systems and their architecture. Many of these formalisms explicitly address composition and interoperation issues. There are important systematic connections between these formalisms that will be crucial for combining and interoperating them to achieve much more flexible and powerful formally based methodologies and tools for developing and evolving complex software systems. In particular, rigorous interoperation of the formalisms will support descriptions spanning multiple levels and views of a software system in a coherent and mathematically rigorous way.

The formalisms that we have discussed can be grouped into three categories: *design notations* such as ADLs, OOD/UML, Graph Grammars (§ 2.1, 2.6, 2.7); *formal specifications,* that come in two flavors: *requirements* specifications and *executable* specifications (§ 2.3, 2.5); and *semantic models* for concurrent object systems and components of open distributde systems (§ 2.4).
We have seen that there is already substantial existing work relating these formalisms. For example:

- the work of the ARTS project mapping engineering notations to formal theories
- the work of Haeberer, Maibaum and others on Fork Algebras
- the work of Meseguer and Talcott mapping ADLs to rewriting logic
- the work of Wirsing and Knapp mapping OOD notations to executable rewriting logic specifications
- the work of Lechner and Denker relating temporal logic requirements to executable rewriting logic specifications
- the work of Duarte relating temporal logics and actors
- the work of Maibaum and Fiadeiro relating architecture to temporal logic
- the work of Meseguer relating graph grammars to rewriting logic,

Furthermore, the capability of rewriting logic as a reflective metalogical framework provides the ability to represent these formalisms and their relations formally. For experimentation and tool building purposes these representations can be made executable using the Maude language.

The U.S. and Brazilian partners bring highly synergistic and complementary expertise to the proposed project. On the U.S. side we can mention reflective metalogical frameworks, experience with formal semantics and interoperation of ADLs, executable formal object-oriented specification, the Maude language, and specification and semantics for the actor model of open distributed systems. On the Brazilian side we can mention temporal logic formalisms and their relation to architecture and formal object-oriented specification, the ARTS project and the integration of OOD notations with executable formal specification and code generation, and graph grammar and Petri net formalisms for open distributed systems.

It is precisely this wide spectrum of expertise and the shared goal of achieving formal interoperation in an architectural setting that allows us to attack the ambitious project of developing a mathematical foundation and carrying out substantial case studies to validate this foundation for methodological purposes.

3 Research Plan

We propose a 3-year collaborative project. We first briefly discuss the organization of the collaboration and give a time line for the work. Then we describe research plans for work on foundations, and for demonstrating the ideas through a set of experiments with specific formalisms.

3.1 Objectives and Approach

The main objectives are (1) developing mathematical foundations for integrating and interoperating in a coherent and mathematically rigorous way diverse architectural descriptions, formal specifications, and executable prototypes of object-based distributed systems, and (2) based on such foundations developing a methodology for system design, development, and evolution that supports a seamless integration of the different formal and informal system descriptions and makes explicit the constraints that these different views of a system impose on each other.

The development of mathematical foundations will begin with an in-depth study of a wide range of notations and formalisms and of their mutual relationships that seem particularly promising for describing systems at different levels. The formalisms chosen to populate and explore the formal architectural space will include semantic models, logics, high-level specification notations, calculi, and executable languages. Some of the specific formalisms that we will consider for instantiating the framework are rewriting logic (with the Maude tool), OMT/UML notations, Community, object calculi, fork algebra, temporal formalisms such as the interval temporal logic of Lano, the temporal logic for actors of Duarte, and the distributed temporal logic of Denker, actor theories and algebras, specification diagrams, transition systems, and interaction models including event structures, event partial orders, and signal models. We will use executable specification languages such as Maude or Community rather than standard programming languages such as C++ or Java to make it feasible to carry out the planned experiments in a 3-year time frame. Further development of the framework would include these languages, along with formal semantics (for usable fragments) and formal (semantics preserving) methods for transforming specifications to efficient code. Rewriting logic will play an important role, not only as an executable specification formalism, but also as a reflective metalogical framework in which the chosen formalisms and their relations can be formalized. These formalizations can be executed using the meta tools developed in the Maude language, an implementation of rewriting logic.

The development of a methodology based on these foundations will be driven by substantial case studies in which the ensemble of formalisms and their systematic relationships will be applied to and tested on concrete applications that demonstrate the interoperation of the formalisms and the subsequent benefits. One case study that we are considering is the Java Card (described below). A second case study will be determined during the project.

3.1.1 Java Card Case Study

The object of the first case study is the software architecture related to Java Cards. Java Cards are SmartCards able to run Java programs. They include a processor and a limited amount of memory in which the operating system is located together with a Java Virtual Machine and a run-time environment.

Java Cards are already present in Industry; they have an immense potential and their application is in a stage of very fast diffusion and growth. The Java Card Forum (see http://www.javacardforum.org) is an organization whose objective is *to promote Java as the preferred programming language for multiple-application SmartCards*.

Java Cards and related issues are an area of big interest for the application of formal methods. The software embedded in the card is naturally mobile, and its correctness is a critical aspect due to its embedded character. Moreover, the applications involving SmartCards are concurrent and distributed, and the market tendency and the speed of hardware evolution make the heterogeneity of the platform an indispensable requirement.

The hardware evolution for Java Cards make consistency and interoperability primary objectives; a Java program must run in different platforms, from Java Cards to workstations, maintaining the same observable behavior. Every unit in a system involving Java Cards, including card (often of different models and technology throughout the life cycle of the system), card readers, communication networks, and central processing equipment, must work harmoniously. Moreover, the whole must allow in its evolution behavioral modifications. Such evolution is induced by technology developments, for instance inclusion of new equipment, and software and hardware updates. For almost two years, LMF-DI has been interested in SmartCards, its applications and related areas. In [43] a survey and diagnosis of the current situation of software development for Java Card is carried out, and aspects where formal/mathematical methods could be applied and could have a technologically significant impact are identified. These aspects include the formal specification of the Java Virtual Machine, of the Java Card operating system and run-time environment, and of the instruction set of the microprocessors (microcontrollers) which are embedded in Java Cards. Also included are the correct-by-construction implementation of Java Virtual Machine and Java Card operating systems and run-time environment, and the conception and construction of an environment for Java program development and byte code generation for Java Card embedded virtual machine.

The figure 2 extracted from [43] shows a proposal for the software development process for Java Card conceived at LMF-DI. Java programs in the programming plane (rear plane) are implemented in Java Card in the foreground plane. The formal plane (the intermediate plane) is used for establishing the relationship between the program and its implementation in the Java Card. In this plane, formal proofs can be carried out.

The design in figure 1 allows better understanding of the diagram in the figure 2. An instantiation of the figure 1 with the information of the figure 1 shows that the components in the formal plane and the programming plane are natural inhabitants of the linguistic plane in figure 1. The denotation of these components into operational models in the semantic plane does not appear explicitly in figure 1. What appear there are the realizations of these operational models in the Java Card, which by nature are not formal. The adequacy of these realizations relative to the programs will be given by the observational equivalence between such realizations and the operational models.

3.2 Organization

The research will be carried out as a U.S.-Brazilian collaboration between researchers at Stanford and SRI in the U.S. and the Laboratory for Formal Methods (LMF) at Pontifical Catholic University of Rio de Janeiro (PUC-Rio) in Brazil in collaboration with researchers at Universidade Federal do Rio Grande do Sul (UFRGS).

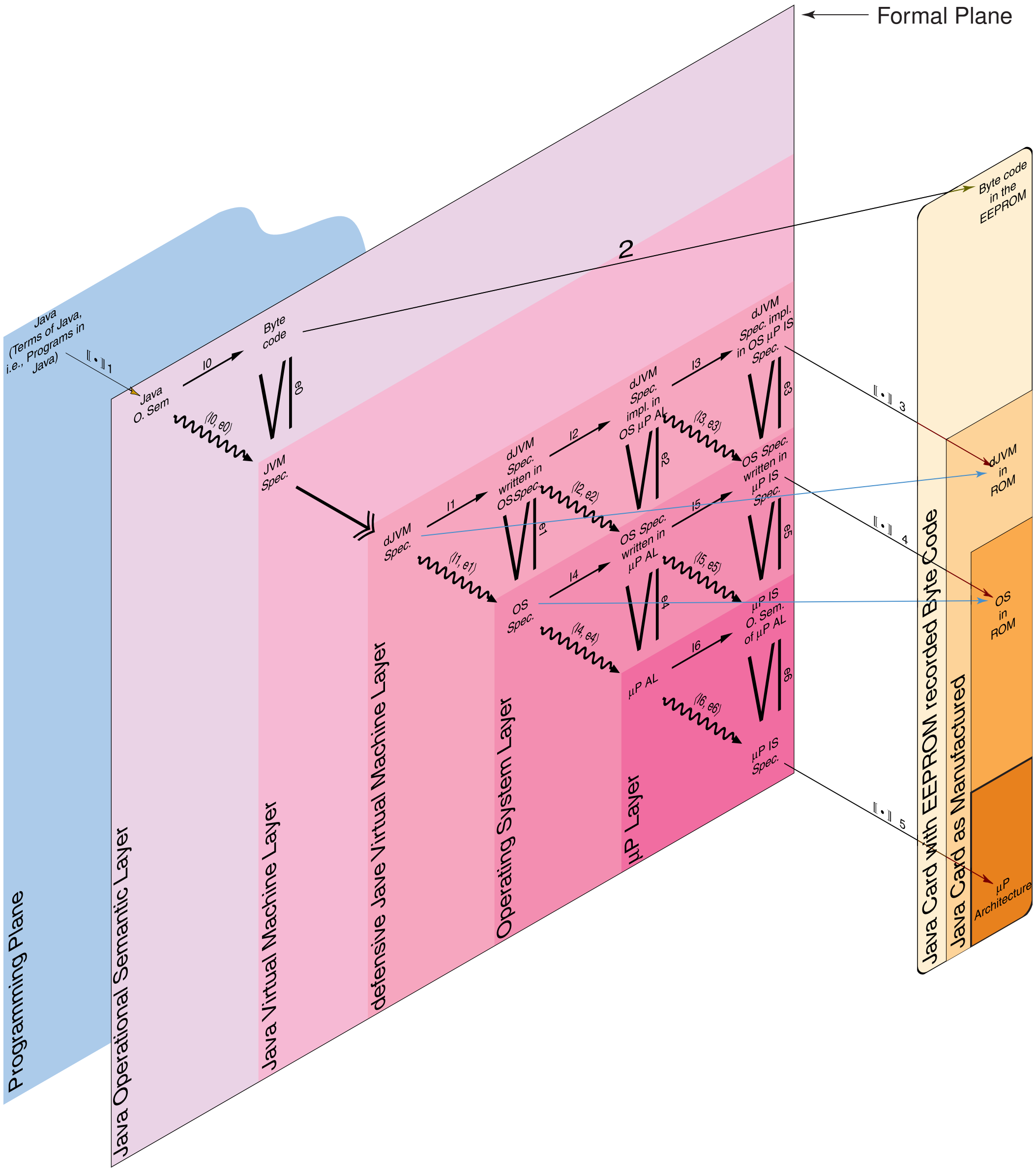

Figure 2: Java Card Analysis

Dr. José Meseguer will be the PI at SRI, and Dr. Carolyn Talcott will be the PI at Stanford. At SRI, Meseguer will be assisted by Dr. Steven Eker, who will help in all aspects of the research, including case studies and prototyping. At Stanford, Talcott will be assisted by a Ph.D. student, yet to be determined, who will help in selected aspects of the theoretical research and in carrying out the case studies.

The PI for the Brazilian group will be Prof. Dr. Armando Haeberer of LMF assisted by a PhD student to be determined and others from the LMF and other collaborating universities in Brazil.

We plan to have two working meetings the first year and one meeting each of the following years, alternating in location between the U.S. and Brazil. Working meetings will include PIs along with key associate researchers and PhD students involved in the project. There will be an exchange visit of 2 to 3 months visit by Ph.D. students of the U.S. and Brazilian partners. In addition, the collaboration will be aided by frequent email and telephone exchanges, and shorter visits by senior researchers as appropriate.

All of the PIs involved are experienced in international collaboration and understand that initial intense real-time interaction is crucial, especially for bringing together the diverse base of expertise to develop new foundational ideas and plan for their application.

The effort will complement and benefit from ongoing joint projects of Meseguer and Talcott related to use of rewriting logic to support development of formal methods and tools. In addition, the effort will benefit from continued collaboration of Profs. Tom Maibaum of Imperial College, London, and José Fiadeiro of University of Lisbon, with the LMF, SRI and Stanford researchers.

3.3 Time line

```
        Year 1                         Year 2                         Year 3

|-----------------------|-----------------------|----------------------|

|----------------------------------------------------------------------
 Mathematical framework
            |----------------------------------------------------------
           Instantiation
                        |----------------------
                         Java Card case study
                                               |-----------------------
                                                  2nd case study

    WM1        WM2             WM3                    WM4
         TR1        TR2                        TR3                      TR4
```

Figure 3: Time Line.

Figure 3 show a time line of the proposed research activities. In more detail the overall plan is as follows.

WM1 1st working meeting 2 months after project start (in Palo Alto, California)

Year 1 6 months study of existing formalisms and development of mathematical foundation formalizing the objects and relations of the framework initiated by WM1

TR1 Technical report detailing first-level understanding of the big picture and detailed plans for future work

WM2 2nd working meeting, 6 months after the first (in Brazil)

Year 1.5 Instantiation of the framework with specific notations and formalisms and working out the formal relations, initiated by WM2

TR2 Technical report describing the results of the instantiation effort

WM3 year 2 working meeting – early to mid year

Year2 Java Card case study

TR3 Technical report describing results of the first case study and its methodological implications

WM4 year 3 working meeting - early to mid year

Year2 2nd case study - to be determined

TR4 Technical report describing results of the second case study and its methodological implications

TR5 Summary report describing results of the overall collaboration effort

4 Results of Prior NSF support

As this is a bundled proposal from SRI International and Stanford University, we include information on results from both principal investigators and an ongoing joint NSF project.

4.1 Meseguer

NSF award number: CCR-9224005
Total Amount: $128,530
Period: 8-15-93 to 7-31-96
Title: Multiparadigm Declarative Programming

The theme of this project, now in its last months, has been the development of metalogical foundations for multiparadigm declarative languages with wide-spectrum support for formal specification, program transformations, and programming, including parallel programming. In the general theoretical framework so developed, the multiparadigm aspects of one such language, Maude, based on rewriting logic have been studied in detail, and implementation techniques, including parallel ones, have been developed. These studies have served as a basis for a prototype compiler mapping a subset of Maude into parallel machine instructions, and for a sequential Maude implementation currently under way.

The metalogical foundations for declarative specification and programming languages have been developed in a series of papers [55, 56, 75, 13, 20] that use the theory of general logics as a metatheory in which the logic of choice for a language can be varied while important modularity and parameterization properties are preserved. The papers [55, 56] also explore the role of rewriting logic and Maude as a logical framework in which many other logics can be represented, and as a semantic framework in which to specify and prototype many languages and systems, including parallel ones. The papers [75, 13] give foundations and make specific suggestions for achieving a much greater degree of interoperation between declarative languages. The paper [20] gives metalogical foundations for *reflective* declarative languages. At the foundational level, categorical models for concurrency have also been studied in two papers [77, 78].

Using these foundations, we have advanced all aspects of the Maude multiparadigm language, which is based on rewriting logic. The paper [55] showed that both equational logic and Horn logic with equality are naturally embedded, in a conservative way, within rewriting logic. At the language-design level, this is reflected in the inclusion of the OBJ equational language as Maude's functional sublanguage, and means that both functional and Horn clause computations can be naturally supported in Maude. In addition, Maude's object-oriented modules support very natural declarative programming of object-oriented systems, including concurrent ones. Therefore, Maude achieves a rigorous and natural integration of the functional, Horn, and object-oriented paradigms. The wide-spectrum aspects of Maude and its support for multiparadigm parallel programming have been studied in [50], where program transformation techniques to pass from specifications to efficient parallel programs are also presented. The way in which Maude, by being based on a logic of change, allows formal executable specification and programming of systems, avoiding altogether the "frame problem," has been explained in detail in [59]. The type-theoretic foundations of OBJ and Maude, as well as a possible higher-order extension of their type theory, have been studied in [57, 54].

The advances obtained in Maude's semantics and language design have served as a basis for implementation work, both in the parallel implementation of Maude and in a sequential implementation. The paper [51] gives compilation techniques for efficiently executing the Simple Maude subset in SIMD and MIMD/SIMD parallel machines, and reports on a prototype compiler mapping rewrite rules onto the Rewrite Rule Machine parallel instruction set. The performance estimates, obtained in a separate project, are very encouraging and demonstrate the practical usefulness of declarative parallel programming in Maude. Since last year, we have also been working on a sequential implementation for Maude that will have advanced features such as powerful module composition operations, a very expressive type theory, user-definable computational strategies, sophisticated equational pattern matching, and reflective and multiparadigm capabilities.

4.2 Talcott

NSF award number: CCR-9312580
Total Amount: $72,973
Period: 6-1-95 to 5-31-97
Title: Reasoning About Open Systems

In [90] a general two-level model of open distributed computation is presented that represents application and runtime activities in a uniform framework. This model establishes a formal basis for specifying and reasoning about resource management activities, dynamic system modifications, and properties such as noninterference and composability. The application level of the model refines the model of [2, 3], explicitly representing runtime structures, location, and allocation of resources. It also abstracts from the choice of specific programming language, providing a framework for reasoning about heterogeneous systems. Applications of this architecture include formal verification of a distributed garbage collection algorithm for actors, and representation of dependability protocols and other runtime aspects of an application as meta-level programs that are independent of the base application behavior. The latter formalizes the ideas presented in [1]. A case study illustrating reasoning about composition of meta-level systems implementing protocols for migration and distributed snapshot is presented in [91].

In [86] two important notions are introduced: *abstract actor structures*, and *interaction semantics*. An abstract actor structure characterizes the minimal semantic requirements for an actor language and allows for components to be defined using multiple languages. It also provides a framework for high-level, programming-language-independent specification of individual actor behaviors. Interaction semantics abstracts away from the details of a component's internal activity and is a basis for reasoning about safety, liveness, and noninterference properties of open components. Interaction semantics is compositional, and semantic properties of combinations of components can be computed from properties of the individual components. This result is the basis of a calculus for composition and transformation of components. The interaction semantics of a component is determined by the abstract actor specification of the individual actors. This provides language independence and greatly simplifies the reasoning.

4.3 Talcott-Meseguer

NSF award numbers: CCR-9633419 and CCR-9633363
Total Amount: $\$420,000$
Period: 89/6-8/99
Title: Semantic Interoperation of Open Systems

Objective This is a joint project between Meseguer and Talcott, funded jointly by NSF and DARPA, but separately budgeted. The objective of this proposed research is to develop a semantic foundation for composition and interoperation of heterogeneous components of open systems. Composition and interoperation occur at many levels (requirements, specification, code, executables) and along multiple dimensions (functionality, reliability, security, availability). Relations among the various levels and dimensions must be preserved. Existing work on composition and interoperation has largely focused on syntactic structure. It is essential to develop a semantic basis to provide for safe, secure, meaningful combinations. Semantic structuring of complex systems should provide a firm basis for determining what parts are interchangeable under given conditions, and to predict effects of changes: of components on the whole, and of system requirements on requirements for the parts.

Recent Accomplishments A family of composable models for components of actor-like systems has been developed spanning a wide range of levels of abstraction from detailed operational semantics to highly abstract interaction path semantics. This has been used in developing methods for reasoning about composition of components and as a semantic basis for specifications.

A general partial-order of events model for object-based Maude specifications has been developed. This provides a semantic basis for interoperation of components described in languages with different underlying computation models.

A preliminary design for a mapping from Wright to Maude has been developed and demonstrated with examples of Wright components represented in Maude. The Wright-to-Maude mapping gives executable semantics to Wright architecture descriptions (prototyping capability), as well as opening up possibilities of new event-based analyses, and semantically sound interoperation with architectures described in other languages.

Specification Diagrams, a diagrammatic notation for specifying actor system components (object-based components of open distributed systems), has been developed in collaboration with researchers at Johns Hopkins University. The notation is similar in spirit to some of the event/message diagram notations of UML, and has a rigorous semantic foundation for composition, transformation, and checking satisfaction.

A systematic mapping of the concurrency models of tile logic (that cover a wide variety of synchronization, coordination, and mobility mechanisms) into rewriting logic has been developed along with methods for executing tile models in Maude through this mapping.

During FY98 the Maude implementation has been greatly advanced, with a beta release in the first quarter of 1998 and a first distributable version planned for December 1998. It has very high performance, reaching about 600,000 logical inferences per second on real applications on a Pentium-Pro computer.

The following papers have been published [21, 89, 60, 87, 9, 8, 31, 18, 28, 19, 67, 72, 76, 71, 88, 74, 52, 25, 16, 68, 7, 6, 84]

Ongoing Work Use of the event model for Maude as a basis for mappings between Maude and Rapide and for the use of Rapide tools in combination with Maude architectural specifications will be investigated.

In collaboration with Cheng at U. Michigan, OMT notations will be formalized in Maude and this will be integrated with Cheng's graphical-based tools for construction and manipulation of specifications to provide additional analysis, prototyping, and interoperation capability.

References

[1] G. Agha, S. Frølund, R. Panwar, and D. Sturman. A linguistic framework for dynamic composition of dependability protocols. In *Proceedings of the 3rd IFIP Working Conference on Dependable Computing for Critical Applications*, September 1992.

[2] G. Agha, I. A. Mason, S. F. Smith, and C. L. Talcott. Towards a theory of actor computation. In *The Third International Conference on Concurrency Theory (CONCUR '92)*, volume 630 of *Lecture Notes in Computer Science*, pages 565–579. Springer Verlag, August 1992.

[3] G. Agha, I. A. Mason, S. F. Smith, and C. L. Talcott. A foundation for actor computation. *Journal of Functional Programming*, 7:1–72, 1997.

[4] R. Allen and D. Garlan. Formalizing architectural connection. In *Proceedings 16th International Conference on Software Engineering*, 1994.

[5] G. Baum, M. Frias, and T. S. E. Maibaum. A logic for real-time system specification, its algebraic semantics and equational calculus. In *Seventh International Conference on Algebraic Methodology and Software Technology (AMAST'98)*, LNCS. Springer-Verlag, 1998.

[6] Adel Bouhoula, Jean-Pierre Jouannaud, and José Meseguer. Specification and proof in membership equational logic. To appear in *Theoretical Computer Science*.

[7] Adel Bouhoula, Jean-Pierre Jouannaud, and José Meseguer. Specification and proof in membership equational logic. In M. Bidoit and M. Dauchet, editors, *Proceedings TAPSOFT'97*, volume 1214 of *Lecture Notes in Computer Science*, pages 67–92. Springer-Verlag, 1997.

[8] R. Bruni, J. Meseguer, and U. Montanari. Process and term tile logic. Technical Report SRI-CSL-98-06, SRI International, July 1998.

[9] Roberto Bruni, José Meseguer, and Ugo Montanari. Internal strategies in a rewriting implementation of tile systems. *Proc. 2nd Intl. Workshop on Rewriting Logic and its Applications*, ENTCS, North Holland, 1998.

[10] L. Cardelli and A. D. Gordon. Mobile ambients. In *Proc. of Foundations of Software Science and Computation Structures (FoSSaCS)*, 1998. European Joint Conferences on Theory and Practice of Software (ETAPS'98), Lisbon, March 30 - April 3, 1998. (To appear).

[11] S. Carvalho. The DDL programming language. Technical report, Department of Informatics, PUC-Rio, Brasil, 1996.

[12] S. Carvalho, J.L. Fiadeiro, and E. H. Haeusler. A formal approach to real-time object-oriented software. In *22nd IFAC/IFIP Workshop on Real Time Programming, Lyon, France*, 1997.

[13] Maura Cerioli and José Meseguer. May I borrow your logic? (Transporting logical structure along maps). *Theoretical Computer Science*, 173:311–347, 1997.

[14] K. M. Chandy and J. Misra. *Parallel Program Design*. Addison-Wesley, 1988.

[15] M. Clavel, S. Eker, P. Lincoln, and J. Meseguer. Principles of Maude. In *Rewriting Logic Workshop'96*, 1996.

[16] Manuel Clavel. Reflection in general logics and in rewriting logic, with applications to the Maude language. Ph.D. Thesis, University of Navarre, 1998.

[17] Manuel Clavel. *Reflection in General Logics, Rewriting Logic, and Maude*. PhD thesis, University of Navarre, 1998.

[18] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and José Quesada. Maude as a metalanguage. *Proc. 2nd Intl. Workshop on Rewriting Logic and its Applications*, ENTCS, North Holland, 1998.

[19] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, and José Meseguer. Metalevel computation in Maude. *Proc. 2nd Intl. Workshop on Rewriting Logic and its Applications*, ENTCS, North Holland, 1998.

[20] Manuel Clavel and José Meseguer. Axiomatizing reflective logics and languages. In Gregor Kiczales, editor, *Proceedings of Reflection'96, San Francisco, California, April 1996*, pages 263–288. Xerox PARC, 1996.

[21] Manuel Clavel and José Meseguer. Internal strategies in a reflective logic. In B. Gramlich and H. Kirchner, editors, *Proceedings of the CADE-14 Workshop on Strategies in Automated Deduction (Townsville, Australia, July 1997)*, pages 1–12, 1997.

[22] G. Denker. From rewrite theories to temporal logic theories. In C. Kirchner and H. Kirchner, editors, *2nd International Workshop on Rewriting Logic and its Applications, WRLA'98*, volume 15 of *Electronic Notes in Theoretical Computer Science*, 1998. URL: `http://www.elsevier.nl/locate/entcs/volume15.html`.

[23] G. Denker and H.-D. Ehrich. Specifying Distributed Information Systems: Fundamentals of an Object-Oriented Approach Using Distributed Temporal Logic. In H. Bowman and J. Derrick, editors, *Formal Methods for Open Object-Based Distributed Systems (FMOODS'97), Volume 2, IFIP TC6 WG6.1 Intern. Workshop, 21-23 July, Canterbury, Kent, UK*, pages 89–104. Chapman & Hall, 1997.

[24] G. Denker, J. Meseguer, and C. Talcott. Protocol specification and analysis in Maude. In *Workshop on Formal Methods and Security Protocols*, June 1998.

[25] G. Denker, J. Meseguer, and C. Talcott. Protocol specification and analysis in maude. In N. Heintze and J. Wing, editors, *Workshop on Formal Methods and Security Protocols, 25 June 1998, Indianapolis, Indiana*, 1998.

[26] C. H. C Duarte. A proof-theoretic approach to the design of object-based mobility. In H. Bowman and J. Derrick, editors, *Formal Methods for Open Object-based Distributed Systems, Volume 2*, pages 37–53. Chapman & Hall, 1997.

[27] C. H. C Duarte. Towards a proof-theoretic foundation for actor specification and verification. In P.-Y. Schobbens and A. Cesta, editors, *4th Workshop on Formal Models of Agents (ModelAge'97)*, 1997.

[28] Francisco Durán and José Meseguer. An extensible module algebra for Maude. *Proc. 2nd Intl. Workshop on Rewriting Logic and its Applications*, ENTCS, North Holland, 1998.

[29] H.-D. Ehrich. Object Specification. Informatik-Bericht 96–07, TU Braunschweig, 1996.

[30] H.-D. Ehrich, C. Caleiro, A Sernadas, and G. Denker. Logics for Specifying Concurrent Information Systems. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*, pages 167–198. Kluwer Academic Publishers, 1998.

[31] Steven Eker. Term rewriting with operator evaluation strategy. *Proc. 2nd Intl. Workshop on Rewriting Logic and its Applications*, ENTCS, North Holland, 1998.

[32] J. Fiadeiro and A. Lopes. Semantics of architectural connectors. In M. Bidoit and M. Dauchet, editors, *TAPSOFT'97: Theory and Practice of Software Development*, volume 1214 of *Lecture Notes in Computer Science*, pages 505–519, 1997.

[33] J. Fiadeiro, A. Lopes, and T. Maibaum. Synthesising interconnections. In R. Bird and L. Meertens, editors, *IFIP TC 2 Working Conference on Algorithmic Languages and Calculi*, pages 240–264. Chapman Hall, 1997.

[34] J. Fiadeiro and T. Maibaum. Interconnecting formalisms: Supporting modularity, reuse, and incrementality. In G. E. Kaiser, editor, *3rd Symposium on Foundations of Software Engineering*, pages 72–80. ACM Press, 1996.

[35] J. Fiadeiro and T. Maibaum. A mathematical toolbox for the software architect. In *8th International Workshop on Software Specification and Design*, pages 46–55. IEEE Computer Society Press, 1996.

[36] J. Fiadeiro and T. Maibaum. Categorical semantics of parallel programming. *Science of Computer Programming*, 28(2–3):111–138, 1997.

[37] J. Fiadeiro, C. Sernadas, T. Maibaum, and G. Saake. Proof-theoretic semantics of object-oriented specification constructs. In R. A. Meersman, W. Kent, and S. Khosla, editors, *IFIP WG 2.6 Working Conference on Object-Oriented Databases: Analysis, Design, and Construction*, pages 243–284. North Holland, 1991.

[38] J. Fiadeiro and M. Wermelinger. Connectors for mobile programs. *IEEE Transactions on Software Engineering*, 24(5):331–341, 1998.

[39] J. Fiadeiro and M. Wermelinger. Towards an algebra of architectural connectors: a case study on synchronisation for mobility. In *9th International Workshop on Software Specification and Design*, pages 133–142. IEEE Computer Society Press, 1998.

[40] M. Fowler and K. Scott. *UML Distilled*. Addison-Wesley, 1997.

[41] N. Francez and I. Foreman. *Interacting Processes*. Addison-Wesley, 1996.

[42] A. Haeberer and T. S. E. Maibaum. The very idea of software development environments: A conceptual architecture for the arts environment paradigm. In *International Conference on Automatic Software Engineering ASE'98*, 1998. to appear.

[43] A.M. Haeberer, J.V.Z. Echague, and M.S. Ferreira. Formal manifold analysis and proposed support of different phases of the java card life cycle, July 1998. LMF-DI Internal Report.

[44] A.M. Haeberer, M. Frias, G. Baum, and P. Veloso. Fork algebras. In C. Brink, W. Kahl, and G. Schmidt, editors, *Relational Methods in Computer Science*, Advances in Computing Science. Springer-Verlag, 1997.

[45] E. H. Haeusler, A. Haeberer, T. Maibaum, and J.L. Fiadeiro. ARTS: A formally supported environment for object-oriented software development. In *ECOOP98*, 1998.

[46] ITU-T. *Recommendation X.903 — ISO/IEC International Standard 10746–3: "ODP Reference Model: Prescriptive Model"*. ISO, 1995.

[47] C. Kirchner and H. Kirchner, editors. *2nd International Workshop on Rewriting Logic and its Applications, WRLA'98*, volume 15 of *Electronic Notes in Theoretical Computer Science*, 1998. URL: `http://www.elsevier.nl/locate/entcs/volume15.html`.

[48] U. Lechner. Object-oriented specification of distributed systems in the μ-calculus and Maude. In J. Meseguer, editor, *Proc. First Intl. Workshop on Rewriting Logic and its Applications*, volume 4 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 1996. URL: `http://www.elsevier.nl/locate/entcs/volume4.html`.

[49] U. Lechner. *Object-oriented specification of distributed systems*. PhD thesis, University of Passau, 1997.

[50] Patrick Lincoln, Narciso Martí-Oliet, and José Meseguer. Specification, transformation, and programming of concurrent systems in rewriting logic. In G.E. Blelloch, K.M. Chandy, and S. Jagannathan, editors, *Specification of Parallel Algorithms*, pages 309–339. DIMACS Series, Vol. 18, American Mathematical Society, 1994.

[51] Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Livio Ricciulli. Compiling rewriting onto SIMD and MIMD/SIMD machines. In *Proceedings of PARLE'94, 6th International Conference on Parallel Architectures and Languages Europe*, pages 37–48. Springer LNCS 817, 1994.

[52] Patrick Lincoln and José Meseguer. Strategic reflection. In B. Gramlich and F. Pfenning, editors, *Proceedings of the CADE-15 Workshop on Strategies in Automated Deduction (Lindau, Germany, July 1998)*, pages 3–9, 1998.

[53] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag, 1991.

[54] Narciso Martí-Oliet and José Meseguer. Inclusions and subtypes II: Higher-order case. To appear in *J. Logic and Computation*; appeared also as SRI Report SRI-CSL-90-16.

[55] Narciso Martí-Oliet and José Meseguer. Rewriting logic as a logical and semantic framework. Technical Report SRI-CSL-93-05, SRI International, Computer Science Laboratory, August 1993. To appear in D. Gabbay, ed., *Handbook of Philosophical Logic*, Kluwer Academic Publishers.

[56] Narciso Martí-Oliet and José Meseguer. General logics and logical frameworks. In D. Gabbay, editor, *What is a Logical System?*, pages 355–392. Oxford University Press, 1994.

[57] Narciso Martí-Oliet and José Meseguer. Inclusions and subtypes I: First order case. *J. Logic and Computation*, 6:409–438, 1996.

[58] Narciso Martí-Oliet and José Meseguer. Rewriting logic as a logical and semantic framework. In D. Gabbay, editor, *Handbook of Philosophical Logic*. Kluwer Academic Publishers, 1997.

[59] Narciso Martí-Oliet and José Meseguer. Action and change in rewriting logic. In R. Pareschi and B. Fronhoefer, editors, *Dynamic Worlds: From the Frame Problem to Knowledge Management*. 1998. To be published by Kluwer Academic Publishers.

[60] I. A. Mason and C. L. Talcott. A semantically sound actor translation. In *ICALP'97*, number 1256 in Lecture Notes in Computer Science, pages 369–378, 1997.

[61] P. B. Menezes and J. F. Costa. Compositional refinement of concurrent systems. *Journal of the Brazilian Computer Society*, 2(1):50–67, 1995. Special Issue on Parallel Computation.

[62] J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.

[63] J. Meseguer, editor. *Proc. 1st Intl. Workshop on Rewriting Logic and its Applications*, number 4 in Electronic Notes in Theoretical Computer Science. Elsevier, 1996. URL: `http://www.elsevier.nl/locate/entcs/volume4.html`.

[64] J. Meseguer. Rewriting logic as a semantic framework for concurrency: a progress report. In *Concur96*, 1996.

[65] J. Meseguer and C. Talcott. Semantic interoperation of dynamic heterogeneous architectures, 1997. Technical presentations to EDCS Architecture Cluster Meeting, April and July 1997.

[66] J. Meseguer and C. Talcott. Formal foundations for compositional software architectures, 1998. Position paper for Workshop on Compositional Software Architectures, Monterey, CA. January 1998.

[67] J. Meseguer and C. Talcott. Mapping OMRS to Rewriting Logic. In C. Kirchner and H. Kirchner, editors, *2nd International Workshop on Rewriting Logic and its Applications, WRLA'98*, volume 15 of *Electronic Notes in Theoretical Computer Science*, 1998. URL: `http://www.elsevier.nl/locate/entcs/volume15.html`.

[68] José Meseguer. A logical framework for distributed systems and communication protocols. *Proc. FORTE/PSTV'98*, Kluwer, 1998.

[69] José Meseguer. Rewriting as a unified model of concurrency. In *Proceedings of the Concur'90 Conference, Amsterdam, August 1990*, pages 384–400. Springer LNCS 458, 1990.

[70] José Meseguer. A logical theory of concurrent objects and its realization in the Maude language. In Gul Agha, Peter Wegner, and Akinori Yonezawa, editors, *Research Directions in Concurrent Object-Oriented Programming*, pages 314–390. MIT Press, 1993.

[71] José Meseguer. Formal interoperability. In *Proceedings of the 1998 Conference on Mathematics in Artificial Intelligence, Fort Laurerdale, Florida, January 1998*, 1998. URL: `http://rutcor.rutgers.edu/ amai/Proceedings.html`.

[72] José Meseguer. Membership algebra as a semantic framework for equational specification. In F. Parisi-Presicce, ed., Proc. WADT'97, 18–61, Springer LNCS 1376, 1998.

[73] José Meseguer. Research directions in rewriting logic. In U. Berger and H. Schwichtenberg, editors, *Computational Logic, NATO Advanced Study Institute, Marktoberdorf, Germany, July 29 – August 6, 1997*. Springer-Verlag, 1998.

[74] José Meseguer. Research directions in rewriting logic. In U. Berger and H. Schwichtenberg, editors, *Computational Logic, NATO Advanced Study Institute, Marktoberdorf, Germany, July 29 – August 6, 1997*. Springer-Verlag, 1998.

[75] José Meseguer and Narciso Martí-Oliet. From abstract data types to logical frameworks. In E. Astesiano, G. Reggio, and A. Tarlecki, editors, *Recent Trends in Data Type Specification, Santa Margherita, Italy, May/June 1994*, pages 48–80. Springer LNCS 906, 1995.

[76] José Meseguer and Ugo Montanari. Mapping tile logic into rewriting logic. in F. Parisi-Presicce, ed., Proc. WADT'97, Springer LNCS 1376, 1998.

[77] José Meseguer, Ugo Montanari, and Vladimiro Sassone. On the model of computation of Place/Transition Petri nets. In *Proceedings 15th International Conference on Application and Theory of Petri Nets*, pages 16–38. Springer LNCS 815, 1994.

[78] José Meseguer, Ugo Montanari, and Vladimiro Sassone. Process versus unfolding semantics for Place/Transition Petri nets. *Theoretical Computer Science*, 153(1–2):171–210, 1996.

[79] Elie Najm and Jean-Bernard Stefani. Computational models for open distributed systems. In H. Bowman and J. Derrick, editors, *Formal Methods for Open Object-based Distributed Systems, Volume 2*, pages 157–176. Chapman & Hall, 1997.

[80] S. Nakajima. Encoding mobility in CafeOBJ: an exercise of describing mobile code-based software architecture. In *Proc. of the CafeOBJ Symposium '98, Numazu, Japan*. CafeOBJ Project, April 1998.

[81] Jim Rumbaugh, Ivar Jacobson, and Grady Booch. *Unified Modeling Language Reference Manual*. Addison-Wesley, 1998.

[82] V. Sassone, M. Nielsen, and G. Winskel. A hierarchy of models for concurrency. In *The Fourth International Conference on Concurrency Theory (CONCUR '93)*, Lecture Notes in Computer Science. Springer Verlag, 1993.

[83] S. F. Smith. Specification diagrams for actor systems. In *Proceedings of Higher-Order Operational Techniques in Semantics, HOOTS II*, volume 10 of *Electronic Notes in Theoretical Computer Science*, 1998.

[84] S. F. Smith and C. L. Talcott. Modular reasoning for actor specification diagrams, 1999. (to appear FMOODS99).

[85] C. L. Talcott. An actor rewriting theory. In J. Meseguer, editor, *Proc. 1st Intl. Workshop on Rewriting Logic and its Applications*, number 4 in Electronic Notes in Theoretical Computer Science, pages 360–383. North Holland, 1996.

[86] C. L. Talcott. Interaction semantics for components of distributed systems. In E. Najm and J-B. Stefani, editors, *1st IFIP Workshop on Formal Methods for Open Object-based Distributed Systems, FMOODS'96*, 1996. Proceedings published in 1997 by Chapman & Hall.

[87] C. L. Talcott. Composable semantic models for actor theories. In M. Abadi and T. Ito, editors, *Theoretical Aspects of Computer Science*, number 1281 in Lecture Notes in Computer Science, pages 321–364. Springer-Verlag, 1997.

[88] C. L. Talcott. Composable semantic models for actor theories. *Higher-Order and Symbolic Computation*, 11(3), 1998.

[89] C. L. Talcott. Reasoning about programs with effects. In *2nd NSF-CNPq Workshop on Semantics*, 1998. to appear Elsevier Electronic Notes in Theoretical Computer Science.

[90] N. Venkatasubramanian and C. L. Talcott. A metaarchitecture for distributed resource management. In *Hawaii International Conference on System Sciences, HICSS-26*, January 1993.

[91] N. Venkatasubramanian and C. L. Talcott. Reasoning about Meta Level Activities in Open Distributed Systems. In *Principles of Distributed Computation (PODC '95)*, pages 144–153. ACM, 1995.

[92] Martin Wirsing and Alexander Knapp. A formal approach to object oriented software engineering. In J. Meseguer, editor, *Proc. 1st Intl. Workshop on Rewriting Logic and its Applications*, number 4 in Electronic Notes in Theoretical Computer Science. North Holland, 1996.

E. Biographical Sketches

- John McCarthy
- Carolyn Talcott
- José Meseguer