



Rapport technique

1993

Open Access

This version of the publication is provided by the author(s) and made available in accordance with the copyright holder(s).

CHASSIS - A Platform for Constructing Open Information Systems

Nierstrasz, Oscar; Konstantas, Dimitri; Dittrich, Klaus; Jonscher, Dirk

How to cite

NIERSTRASZ, Oscar et al. CHASSIS - A Platform for Constructing Open Information Systems. 1993

This publication URL: <https://archive-ouverte.unige.ch/unige:72612>

CHASSIS — A Platform for Constructing Open Information Systems *

Oscar Nierstrasz[†], Dimitri Konstantas[‡]
Klaus Dittrich, Dirk Jonscher^{**}

Abstract

Present-day computer-based information systems are increasingly required to be open systems. This means that they must cope with open networks, heterogeneous interoperable hardware and software systems, and, above all, evolving and changing requirements. The CHASSIS project aims to develop a software and methodology framework for (i) the security- and reliability-oriented systematic design and construction of heterogeneous information systems from individual existing and newly developed application software components and database systems, and (ii) their secure and reliable interoperation. In CHASSIS, object-orientation is the key technology for the construction of such a system as its uniform interface is realized by an object-oriented data model and the homogenization layer is realized by object-oriented software. CHASSIS includes object models for database and language integration, software to support system integration, specification methods to support the design process and advanced security mechanisms to provide the resulting information system with a high degree of security. CHASSIS is a joint Swiss project between the University of Zürich, the University of Geneva, and the Asea Brown Boveri Research Centre (Baden).

Keywords: open systems, security, federated databases, object-orientation.

*.A French version of this paper has appeared as: "CHASSIS — Une Plate-forme pour la Construction de Systèmes d'Information Ouverts," in *Proceedings, AFCET '93 — Vers des Systèmes d'Information Flexibles*, Versailles, June 8-10, 1993, pp. 153-161.

†.Author's current address: Institut für Informatik und angewandte Mathematik (IAM), University of Berne, Länggassstrasse 51, CH-3012 Berne, Switzerland. Tel: +41 (31) 631.4618. E-mail: oscar@iam.unibe.ch. WWW: <http://iamwww.unibe.ch/~oscar>.

‡.Université de Genève, Centre Universitaire d'Informatique, 24 rue Général Dufour, CH-1211 Genève 4, Switzerland. E-mail: {oscar,dimitri}@cui.unige.ch. Tel: +41 (22) 705.7664/7647. Fax: +41 (22) 320.2927.

**.Universität Zürich, Institut für Informatik, Winterthurerstrasse 190, CH-8057 Zürich, Switzerland. E-mail: {dittrich,jonscher}@ifi.unizh.ch. Tel: +41 1 257.4312/4337. Fax: +41 1 363.0035.

1 Introduction

Changes in the hardware and software landscape of computer-based application systems in recent years have led towards distributed, networked and "open" solutions, and away from centralized, proprietary systems. The new systems are open in terms of platform, topology and evolution. Although openness, security and reliability are well-understood at a systems level, this is unfortunately not yet the case for applications software. So a typical user of a modern computer system will find himself (or herself) with a rich variety of software tools, database systems and applications available as "services" through the network, but, in most cases, without any coherent and systematic means to combine these tools to solve everyday problems. Furthermore, it is not presently possible in a distributed, heterogeneous environment to share information, make existing and newly developed software systems interoperable, and construct new information systems based on already available services, data and software, with reasonable effort and in a safe way.

CHASSIS^{††} is a Swiss federal research project of the Programme Prioritaire Informatique that aims to provide a platform for the security- and reliability-oriented systematic design and construction of heterogeneous information systems from individual existing and newly developed application soft-

ware components and database systems. Partners in CHASSIS are the University of Zürich, the University of Geneva, and the Asea Brown Boveri Research Centre (Baden).

Although CHASSIS aims to provide a general-purpose framework, our initial requirements will come from the domain of engineering information systems for electrical engineering (the speciality of ABB). In these application fields many different types of engineering tasks are necessary to build a complete system, such as mechanical CAD, electrical CAD (e.g., wiring diagrams), application-specific calculations (e.g., mechanical stresses, heat transfer, energy efficiency etc.), document preparation, composition (or configuration) of customer-specific systems from standard parts, and parameterization and programming according to customer requirements (e.g. of a process control system). In addition, the various organizational aspects play an increasingly important role, such as collaboration of many teams, often located in different regions, optimization and integration of the engineering process from offer preparation to the final acceptance test, and project management and control.

The key problem that CHASSIS attempts to deal with is the secure integration of heterogeneous, open systems. Traditional software systems are closed in all the ways that modern systems are now required to be open, but the most important of these is that we cannot assume even the system requirements themselves to be closed and fixed. Flexible applications and information systems must be open to changing requirements. CHASSIS specifically targets the problems of (1) secure inte-

††.Configurable, Heterogeneous, And Safe, Secure Information Systems.

gration of heterogeneous, autonomous database systems, and (2) interoperability of heterogeneous software and database systems. CHASSIS adopts an object-oriented approach to both these problems, by introducing (1) an object-oriented data model and corresponding security model for a federated database, and (2) an object-oriented framework for interoperability in which every independent software system and information system can be encapsulated as a *cell*, which contains the original system as its *nucleus*, and adds a *membrane* that is responsible for type matching, object mapping and connection trading. In a later phase of the project, we plan to develop a set of reusable software components for system integration, based on the CHASSIS framework.

2 The CHASSIS Framework

The goal of CHASSIS is to provide for the secure and reliable interoperation of heterogeneous software applications and information systems. To as large a degree as possible, interoperability should not require existing services and systems to be modified. In particular, existing applications should continue to work without alteration when their underlying database systems become available to a network of clients. Furthermore, local autonomy must be preserved, so security must be maintained according to *local* requirements. A global security scheme must take into account differences in policy between constituent database systems.

Looking deeper at the problem, one can distinguish two features that an existing software component can provide for an information system: functionality (services) and data. Consequently, making a software system interoperable means to consider the following two scenarios:

- If just the services it offers are of interest within the integrated information system, it is sufficient to treat it as a black box. It is encapsulated as an object that provides its functionality in a uniform way such that it can be used by others.
- However, as soon as the software component works on some permanent data, it is most likely that interoperability means to make these data also accessible within the information system and to share them with other components. As a consequence, a uniform, integrated view on all the data from the different applications is needed, i.e., as a federated system.

This federated system not only plays a key role in system integration, but takes on special importance with respect to security. On the one hand, it provides the basis for enforcing security in the entire integrated information system and on the other hand, it has to preserve as far as possible the security of participating component database systems. So it must include comprehensive concepts for the protection of data stored in the integrated individual and autonomous database systems (e.g. access and information flow control), since local systems might not be willing to join a federation with weak security properties. In CHASSIS, object-orientation is the key technology for the construction of such a system as its uniform interface is realized by an object-oriented data model and the homogenization layer is realized by object-oriented software.

In the first of the two alternatives sketched above, an application leaves its handling of persistent data unchanged, i.e. the data repository it uses is either isolated in the object representing the application within the integrated information system, or is directly accessed by the application by circumventing the storage service component. In the second case, the application software has to be modified by replacing the parts dealing with permanently storing data by appropriate calls to the federated database system. Although the latter may possibly require considerable programming efforts, it is the only way to effectively share data among applications and benefit from the integration and security features provided by the storage services.

The Federated Database is itself encapsulated within the overall integrated information system (see Figure 1) as a *cell*.

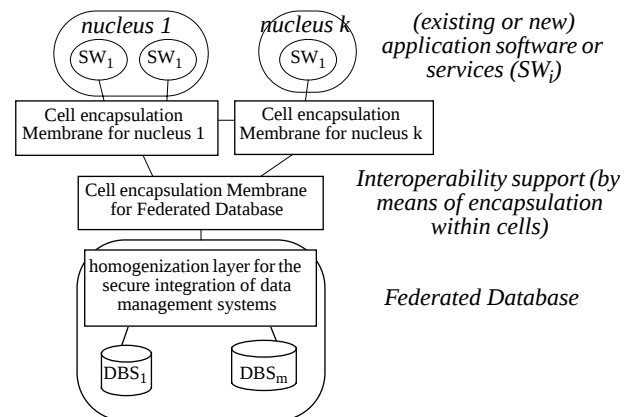


Figure 1 Architecture of an integrated information system based on CHASSIS

The Cell framework provides object-oriented support for the interoperability of heterogeneous software systems at the language level. A cell defines an independent name space and is composed of two parts: a *nucleus* and a *membrane* (see Figure 2). The nucleus is the collection of “user” and system applications (possibly, but not necessarily object-oriented), which we will refer to in general as “objects,” that are responsible for providing services and managing the local resources. The membrane, which surrounds the nucleus, handles all communication with the external world, that is with other remote cells. The objects of the nucleus are not directly visible to other cells and cannot “see” outside the nucleus. It is the membrane that is responsible for the mapping of remote objects to the nucleus (that is to introduce a proxy of the remote object in the local nucleus), and for providing for the transformations of remote types to local ones.

The most important services offered by the membrane of a Cell are *type matching*, *object mapping* and *connection trading*. Type matching is a service that allows the “user” to define a correspondence of a local type to a type or types of the remote type hierarchy, that is a *type match*. Object mapping allows the remote objects to be accessed through the local name space. Connection trading handles the terms under which a remote object is mapped to the local name space. During the mapping of an object to the nucleus of another cell we say that the cells are *connected*.

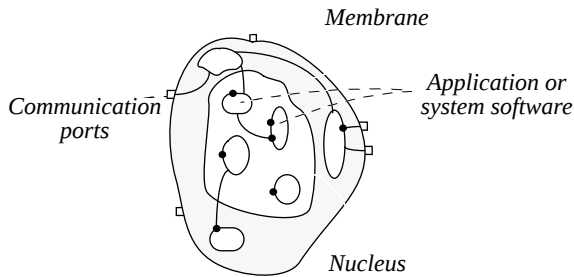


Figure 2 A Cell

In the definition of a type match, the “user” can specify the correspondence between the operation of the local and remote type and the relation of their parameters. Operations can be coerced, parameters can be adapted and new code can be introduced to handle cases where a simple mapping is not possible.

According to the defined type match, objects from remote cells are mapped to the nucleus of the local cell. This way objects from different environments, on remote cells that can be based on different languages or different programming paradigms, can be accessed through a familiar local language and paradigm. With the use of reciprocal object mappings objects of any type can be passed as parameters of an operation.

Before a connection between two cells can be established, the terms of the connection need to be negotiated. These terms include, but are not restricted to, the duration of the connection, the security algorithms that will be used during the exchange of messages, the access rights to the local and remote objects, the charging rates etc.

The Cell framework allows application objects that are implemented on different languages, object-oriented or not, and running on different corresponding execution environments to cooperate and communicate via higher level object types and not only via simple data type objects (integers, strings etc.), which in addition are part of their local execution environment. This way existing software components on remote cells can be reused without any modifications via local proxies that follow the conventions, abstractions and mechanisms of the local environment. As a result the interoperation of applications becomes more reliable and safe since it is based on tested existing software components. Furthermore the Cell framework allows the dynamic (re)configuration of object connections so that applications can be introduced, changed or suppressed at any time without any side effects (overheads, blocking of communication channels etc.) for the running applications.

3 The Federated Database System

As mentioned above, a federation can be considered to be a particular cell of an information system. However, the integration of already existing, autonomous component database systems raises some unique security questions. The key issue in this realm of distributed data processing is the need to offer a unified, transparent view on data which are stored in different databases by preserving local autonomy. The latter covers authorization autonomy as well. We have chosen a tightly coupled system to allow for global access control. Some of the main problems we are faced with stem from heterogeneity (different

data models and different access control mechanisms in component database systems). The challenge is to develop a global security model (based on the global — i.e. object-oriented — data model) and to map global authorizations onto corresponding local ones by preserving local autonomy. Furthermore, the federation allows for establishing inter-database relationships which must be protected, too.

The object-oriented data model at the integration layer basically conforms to the object-oriented database system manifesto [2] and can be characterized as follows: Each entity within the system is an object consisting of a set of (complex) attributes and a set of methods which encapsulate their behaviour. We do not stick to *strict* encapsulation, though. A subset of attributes can be made visible (and accessible) to the public. Objects have an identity independent of their values, i.e. identity is a property that distinguishes objects from each other. They are instances of exactly one type. The inheritance paradigm is based on inclusion and specialization inheritance. We allow for multiple inheritance restricted to strict inclusion inheritance. The chosen message passing paradigm is rather simple. The receiver of a message is always an object and the selector coincides with the name of the method to be invoked. Summing up, the model is close to ZOO/Ifi* [5] which is a companion project at Zürich University aimed at developing an integration framework for database federations. The intention is to enhance ZOO/Ifi by access control functionality.

The global security model developed so far has the following main features [3]: It is based on discretionary access control, i.e., on the identity of the subject and of the protected object. We have chosen a mixed system, providing for positive and negative authorizations, with a closed world assumption (closure assumption in case of incomplete specification). A request is rejected if neither a positive nor a negative authorization can be inferred from the set of explicit authorizations. Furthermore, we apply a simple scheme for conflict resolution. Negative authorizations always override positive ones. Decentralized authorization is based on an ownership paradigm and the well-known “grant options” [4]. Considerations on administration paradigms will follow. We distinguish between two kinds of subjects: users and roles. Roles describe the organizational, functional or social position of users within the universe of discourse and can be used, e.g., to model the structure of companies. Users may play several roles (even at the same time) inheriting their access rights. Activation and deactivation of roles is done explicitly and restricted by conflict relations, i.e. it is possible to prevent the concurrent activation of roles which may result in an insecure accumulation of rights. The system makes heavy use of implicit authorizations. Roles may be in a relationship of subordination (this binary relationship establishes a partial ordered set). Superior roles inherit positive authorizations from their subroles whereas subordinated roles inherit negative authorizations from their superior roles. This results in a system where superior roles have strictly greater power (or authority) than their subordinates. Furthermore,

*.Zürich object-oriented integration framework for building Heterogeneous Database Systems

complex objects are a unit of authorization, i.e. rights for complex objects are inherited by all its component objects. There are some other implications required for object-oriented data models which are not presented here in more detail.

Optionally, a domain concept can be used. We distinguish two different kinds of domains: subject domains and protection object domains. A subject domain consists of a set of users, roles or other subject domains. A protection object domain may include any subset of protection objects, i.e. types, extensions of types, (complex) objects and other protection object domains. They are used for implicit authorizations too. Authorizations for domains (being positive or negative) are inherited by its elements. Domains provide a powerful means for modelling several important security concepts like nested workgroups or private databases.

The global security model will be implemented in an object-oriented manner, too, using ObjectStore (a commercial object-oriented database system) to store these metadata.

The architecture of the global access control system is based on a distributed reference monitor which intercepts messages, evaluates whether they are authorized or not and mediates them to the receiver. An instance of this monitor is running on every node where the federation itself is running. Note that this monitor is only the first line of defence since every local component database system has (theoretically) an access control system of its own.

We do not consider any problem of network security, since this is an independent research direction and we can apply the results already available (e.g., encryption techniques and mutual authentication).

The demonstrator will be implemented with C++ integrating an object-oriented (ObjectStore) and a relational database system (Oracle). As ObjectStore does not offer any access control mechanism, the real challenge is to map the global security concepts onto the local ones of Oracle. The key issue is that local (Oracle) decisions always take priority and that the global model is much more powerful than the access control mechanisms which are locally available. We intend to ensure a consistent authorization state between the global and the local level. The local access control decisions can be based on two different paradigms. The federation has to deliver a trustworthy identity of the global user who has initiated the mediated request at the federation level, or the federation itself appears to be a local user (or to be more precise, a local application). In the latter case, the local systems must put some trust into the security mechanisms of the federation, i.e. they have to sacrifice their local authorization autonomy to a certain degree. However, both solutions are only the extremes of a continuous range of decisions, since the federation may map several global users to (virtual) local users [10]. The scheme which should be used is a matter of negotiation between the global and the local security administrator.

4 Object-Oriented Interoperability

Since applications can exchange information through their common federated database system we can say that a federated database system provides a base for interconnecting the appli-

cations from “below.” However, not all applications use the federated database system, and more than one federated database system can coexist in a network. In addition an application might also need to directly utilize the services offered by other applications even if they are all interconnected with the same federated database system. However in an open environment a given service will be offered by many servers through possibly different interfaces. As a result an application wishing to access a specific service offered from different applications will have to use a different interface for accessing the server of each application. The situation becomes even more complicated if the different applications use different security mechanisms and protocols.

One solution to the above multiple interface problem has been proposed by the Object Management Group (OMG) with the Common Object Request Broker Architecture (CORBA)[1]. However CORBA does not allow flexible transformations of interfaces nor can it handle the exchange of types other than data types and their (simple) aggregates. Furthermore, CORBA does not consider any security issues. A better suited solution for the CHASSIS project is the *Object Oriented Interoperability (OOI)* support [7] of the *Cell framework* [6][8]. This support is based on three services: *Type Matching*, *Object Mapping* and *Connection Trading*.

4.1 Type Matching

Type Matching consists of defining the bindings and transformations from the interface that the client requests to the one offered by the server. These transformations and bindings are expressed in a *Type Matching Specification Language (TMSL)* which is specific to each cell and thus can have a syntax similar to the native programming language of the cell.

Using TMSL the “user” can specify the relations between operations of the local type (interface) to the remote type and how the offered parameters should be transformed to match the requested parameters. TMSL is powerful enough to support not only the definition of the relation between operations but also pre- and post-processing of parameters and results, aggregation and segregation of operations and types, and the definition of adaptation functions for higher flexibility.

In the definition of a type match in we distinguish three kinds of relations between types of the local and the remote type hierarchies: Equivalent, Translated and Type Matched.

Equivalent types are types which exist in both cells with the same semantics and structure. This is most commonly the case with data types, like for example integers, strings and their aggregates. Equivalent types can be migrated from one cell to another without any modification (except possibly modifications of the internal representation, like byte ordering).

Translated types are types which have the same semantics in the two cells but have different structure and representation. For example strings can be represented in one cell as arrays of characters while in another a special type “string” might be present. In this case a string (represented as an array of characters) can be migrated to the second cell with a translation to a “string” object.

Type Matched types are types which have their interfaces linked via a type match relation. This is the case not only for the primary servers of a cell but also for other objects, like console, array processor and database, which cannot be migrated to a remote cell.

4.2 Object Mapping

Whereas type matching maintains the static information of the interoperability templates, object mapping provides the dynamic support and implementation of the interoperability links. We distinguish two parts in object mapping: the static and the dynamic. The static part of object mapping is responsible for the creation of the classes that implement the interoperability links as specified by the corresponding type matching. The dynamic part on the other hand, is responsible for the instantiation and management of the objects used during the interoperability.

The essence of object mapping is to dynamically introduce in the local node the services of servers found on other nodes. This however must be done in such way so that the access of the services is done according to the local conventions and paradigms. In an object oriented node this will be achieved with the instantiation of a local object that represents the remote server, which in OOI we call an *inter-object*. An inter-object is an instance of a type for which a type match has been defined. The class (that is, the implementation of a type) of the inter-object is created by the object mapper from the type match information and we call it *inter-class*. An inter-class is generated automatically by the object mapper and it includes all code needed for implementing the links to the remote server or servers.

After the instantiation of an inter-object and the establishment of the links to the remote server, the controlling application will start calling the operations of the inter-object passing other objects as parameters. OOI allows objects of any type to be used as a parameters at operation calls. The object mapper will handle the parameter objects according to their type relations with the remote node. This way objects whose type has an equivalent or translated one on the remote node, will be migrated, while objects for which a type match exists will be accessed through an inter-object on the remote node.

4.3 Connection Trading

Connection trading is initiated at the time when the a connection is requested. There are two modes under which the Connection Trader operates: *master* and *slave*. The Connection Trader at the client side runs in slave mode, while the one at the server side runs in master mode. The reason for the distinction comes from the fact that the server cell is the one that should impose its terms to the client. If the client does not like the terms of the service it can access a different server!

With the Cell encapsulation the nucleus is relieved from all security considerations from the external world, which are handled by the membrane. Every access to the information system will be preceded by a session of connection trading where the security requirements and other connection parameters will be defined. Once the connection trading is completed successfully the actual services will be provided. Nevertheless, the connection parameters can be dynamically re-negotiate and new secu-

rity procedures can be established or even the services can be interrupted in case a breach of security is suspected. Furthermore even during a service connection different security requirements for specific operations or data exchange might be imposed dynamically via a special connection trading sessions. However all the dynamic negotiation of security procedures will be transparent to the nucleus whose objects will have the notion of offering their services to other local objects.

5 Concluding Remarks

The CHASSIS project aims to address the secure and reliable interoperation of heterogeneous applications and database systems by means of an object-oriented interoperability framework and an object-oriented data model for database federation. The requirements for CHASSIS will initially be provided from the domain of electrical engineering information systems, and the results will be applied in a prototype system in this domain.

The object-oriented framework for interoperability features the notion of a *Cell*, that encapsulates an existing system by surrounding it with a *membrane* responsible for negotiating communication with external applications and services. In the CHASSIS project we plan to study the connection trading requirements and procedures with special focus in the security issues. In addition we will study the security implications that type matching and object mapping have in the interconnection of different information systems.

The object-oriented data model for database federation hides the data models of constituent databases by providing for an integration layer in which a global object-oriented schema can be expressed. A global security model will be developed to cope with the different security requirements and policies of the underlying databases.

Finally, we hope to develop a framework of reusable software components for interconnecting heterogeneous systems in a type-safe and secure fashion. This will be done using a *pattern language* presently being designed as part of another project ("Active and Multimedia Objects"). The pattern language [9] is intended to simplify and generalize object-oriented mechanisms for encapsulation and reuse by introducing active objects as the basic computational entities, and *patterns* as the fundamental abstraction mechanism for developing reusable software components (subsuming classes, inheritance and other related mechanisms). As the pattern language will itself be under development during the initial phase of CHASSIS, work on the development of the framework within CHASSIS will necessarily begin later and be of a more experimental nature.

References

- [1] *The Common Object Request Broker: Architecture and Specification*, Object Management Group and X Open, Document Number 91.12.1 Revision 1.1
- [2] M. Atkinson et al., “The Object-Oriented Database System Manifesto,” Proc. 1st Int. Conf. on Deductive and Object-Oriented Databases, Kyoto, Japan, Dec. 1989
- [3] D. Denning, “Cryptography and Data Security,” Addison-Wesley, Reading Massachusetts, 1982
- [4] P.P. Griffith and B.W. Wade, “An Authorization Mechanism for a Relational Database System,” ACM TODS, Vol. 1, No. 3, Sep. 1976, 242-255
- [5] M. Haertig and K. R. Dittrich, “An Object-Oriented Integration Framework for Building Heterogeneous Database Systems,” in *Proc. of the IFIP DS-5 Conf. on Semantics of Interoperable Database Systems*, Lorne, Australia, Nov. 1992.
- [6] D. Konstantas, “Design Issues of a Strongly Distributed Object Based System,” in *Proceedings of 2nd IEEE International Workshop for Object-Oriented in Operating Systems (I-WOOOS '91)*, Palo-Alto, October 17-18 1991, pp. 156-163.
- [7] D. Konstantas, “Object-Oriented Interoperability,” in *Proceedings ECOOP '93*, ed. O. Nierstrasz, LNCS, Springer-Verlag, Kaiserslautern, Germany, July 1993, to appear.
- [8] D. Konstantas, “Hybrid Cell: An Implementation of an Object Based Strongly Distributed System,” in *Proceedings of the International Symposium on Autonomous Decentralized Systems — ISADS 93*, Kawasaki, Japan, March 30 1993, to appear.
- [9] O. Nierstrasz, “Composing Active Objects — The Next 700 Concurrent Object-Oriented Languages,” in *Research Directions in Concurrent Object Oriented Programming*, ed. G. Agha, P. Wegner and A. Yonezawa, MIT Press, 1993, to appear.
- [10] M. Templeton, E. Lund and P. Ward, “Pragmatics of Access Control in Mermaid,” *Data Engineering*, Vol. 10, No. 3, Sep. 1987 (Special Issue on Federated Database Systems).