4. DESCRIPTION OF THE *ACRU* AGROHYDROLOGICAL MODELLING SYSTEM

The literature review in the previous chapter has revealed that most of the existing hydrosalinity models that are commonly used in South Africa operate only as lumped model. However, catchment based hydrosalinity modelling usually requires discretisation of the catchment into a number of sub-catchments, especially when modelling larger catchments or catchments with complex land use and soils. Therefore, the hydrosalinity module is decided to be developed within the *ACRU* model, since *ACRU* can operate as a distributed cell-type model and with widely tested multi-purpose hydrological modules.

ACRU is a conceptually-physically based agrohydrological modelling system (Schulze, 2001). The model was initially written in FORTRAN 77. However, as will be described in this chapter, this programming language had some drawbacks when applied to modelling the hydrological system. In order to overcome some of these difficulties, and to accommodate future model additions, the model was recently rewritten in an object-oriented framework using Java programming language (Kiker and Clark, 2001). The new object oriented version of *ACRU* is named *ACRU2000*. Thus, in this chapter and subsequent chapters, the *ACRU* model prior to the development of the object oriented version (*ACRU2000*), will be referred to as the *ACRU 300 series*. This chapter commences with a review of the general background and concepts of the *ACRU* model in which the hydrosalinity module is developed. It will also explain some of the structural limitations associated with the *ACRU 300 series,* followed by an overview of object-oriented programming and the *ACRU2000*. The overview of object-oriented programming explains some of the concepts and terminologies to which there will be a frequent mention in the subsequent chapter that deals with development of the hydrosalinity module.

4.1 Background and Concepts of *ACRU* Model

The acronym *ACRU* is derived from the Agricultural Catchments Research Unit in the Department of Agricultural Engineering, now School of Bioresources Engineering and Environmental Hydrology of the University of Natal in Pietermaritzburg, South Africa (Schulze, 1995a). According to Schulze (1995a), agrohydrology seeks to evaluate the

influence of available water on the agricultural potential, with the objective of promoting a high efficiency for the use of the water. Thus, it can be seen not only as a branch, but also as an extension, of the terrestrial hydrological system when it comes to production information for planning and management of water resources in the broader sense. *ACRU* is an agrohydrological modelling system, which integrates the fields of scientific hydrology, applied engineering and water resources related hydrology with subsequent linking of these fields with agrohydrology. The origin of *ACRU* dates back to detailed studies in the Natal Drakensberg in 1975, where it started as an energy driven catchment evapotranspiration model (Schulze, 1975).

ACRU is centered on a number of concepts that characterise the model. First, it is a conceptual-physical model. It is conceptual in that it conceives of a system in which important processes are idealised, and it is physical in that physical processes are represented in the model explicitly. In order to capture relevant processes, the model uses daily time steps and thus uses daily rainfall and reference potential evaporation data as primary inputs. *ACRU* operates on a daily multi-layered soil water budget (Figure 4.1). This enables the model to simulate land use and climate change impacts on the hydrological system of an area. The model also provides multiple options in many of its routines that can be used depending on the level of input data available or the detail of output required. It can operate either as a lumped small catchment model or as a distributed cell-type model for larger catchments or in areas of complex land use and soils.

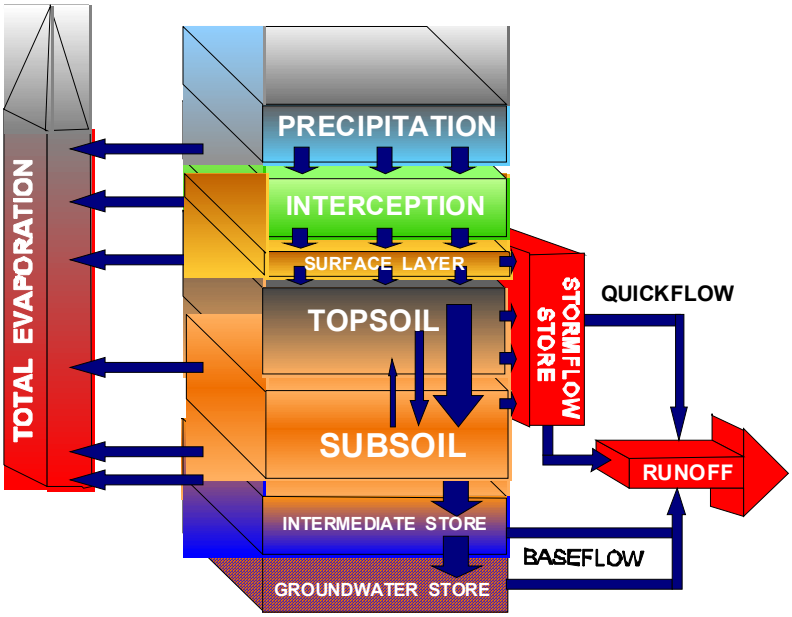

Figure 4.1 General structure of the *ACRU* agrohydrological modeling system (Schulze, 1995a)

ACRU is a multi-purpose model that simulates one or more facets of the terrestrial hydrological system and, according to Schulze (1995a), the model has been widely used by numerous groups of people since 1986, ranging from students to researchers. Typical applications of the model include water resources assessment, design flood estimation, irrigation water requirements and supply, crop yield and primary production modeling, assessment of land use and climate change impacts on water resources, and hydrological impacts of wetlands (Figure 4.2). A detailed explanation on the background, concepts and applications of the *ACRU* model is given by Schulze (1995a) in Chapter 2 of the text accompanying the *ACRU 300* modeling system.

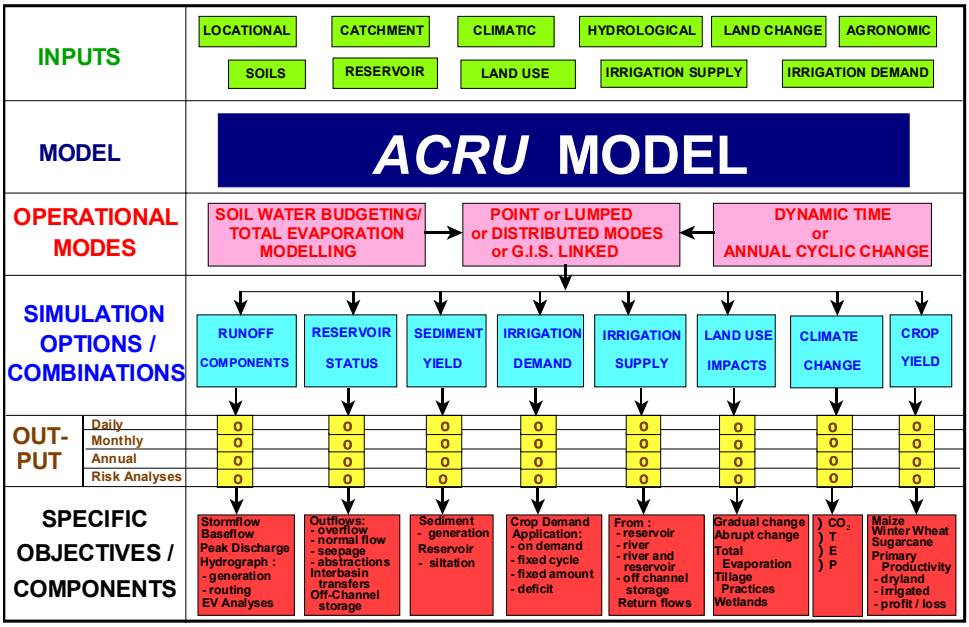

Figure 4.2 The *ACRU* agrohydrological modelling system: Concepts (Schulze, 1995a)

4.2 *ACRU 300 Series* and Its Structural Limitations

The *ACRU 300 series* have been written in the FORTRAN 77 programming language. Although this programming language has many merits in terms of computational efficiency, it also has limitations in developing a modular, easily expandable program design (Campbell *et al.*, 2001) which otherwise could be achieved by any object-oriented programming language such as Java. In his comparison between object oriented and procedural simulation models, such as FORTRAN and BASIC, from the view of their suitability for simulating an ecosystem, Silvert (1993) has described object-oriented simulation models as consisting of objects with complex internal dynamics that interact with each other. Furthermore, he

suggests the existence of a strong resemblance between object-oriented models and the ecosystem, which these models represent, as compared to the resemblance between procedural simulation models and the ecosystem.

Since its inception, *ACRU* has been expanding from being a rainfall: runoff model to its present status with several hundred routines and a complex internal structure. However, according to Campbell *et al.* (2001) it was becoming difficult to make new additions to the model. These structural limitations of the *ACRU 300 series* necessitated a rewriting of the model using an object-oriented programming language and restructuring it into a more extensible and modular structure.

4.3 *ACRU2000*

According to Clark *et al.* (2001) the two main reasons for restructuring the model were to make it easily extendible and to better represent the individual spatial elements of the model and the order of processing to facilitate the modelling of artificial water flows. The restructured *ACRU* model is designated as *ACRU2000*. It is implemented with the help of object-oriented programming techniques and the Java programming language.

Restructuring of the model in general was aimed at satisfying the requirements of both model developers and users. According to Campbell *et al.* (2001), the present modular structure of *ACRU2000* allows different individuals involved in model development to work independently without causing conflicts within the model when their contributions are combined. The structuring of *ACRU2000* was also partially driven by the increased social and governmental interest in water related issues. Various stakeholder groups were requesting new capabilities and tools that could allow them effective management of hydrological information.

4.3.1 Object-oriented programming and the *ACRU2000*

Model development in *ACRU2000* comprises two consecutive steps, object design (including analysis) and the subsequent code development (Campbell *et al.*, 2001). According to Quatrani (1998) the Unified Modeling Language (UML) provides a very robust notation, which grows from analysis into design. UML is a language used to specify, visualise, and

document the artifacts of an object-oriented system under development. UML is widely used in the development of *ACRU2000* for the abovementioned uses. The second step, code development, is implemented with the help of the Java object-oriented programming language. Detailed descriptions of object-oriented programming techniques and the Unified Modeling Language are given in various modelling tools (for example, Quatrani (1998) and Rational Software Corporation (1998)). However, this section is aimed at introducing the reader with some basic concepts and terminologies of object-oriented programming technique, which will be mentioned frequently in the rest of the document.

The object-oriented programming technique is an intuitive way of modelling real world systems such as the hydrological system, in a conceptual manner (Clark *et al.*, 2001). From the modelling point of view, an Object can be described as a concept, abstraction, or matter with well-defined boundaries and meaning for a certain application. A Class is a description of a group of objects with common properties and relationships to other objects and semantics (Quatrani, 1998). Since objects are a representation of either a real world or conceptual entity, there is always an interaction between objects.

Three main relationship types are used in *ACRU2000* to describe interactions between classes or objects, *viz.* inheritance, aggregation and association relationships (Kiker and Clark, 2001). According to Quatrani (1998) and Rational Software Corporation (1998), inheritance defines a relationship among classes where one class shares the structure and / or behavior of one or more other classes. This type of relationship is also called an “is-a” or “ kind-of” hierarchy. In the literature two classes linked by an inheritance are described as child and parent, sub-class and super-class as well as client and supplier (for example, Rational Software Corporation 1998). A child class inherits all attributes, operations, and relationships defined in any of its parent classes. The inheritance relationship is diagrammatically denoted by a line with a triangle at one end connecting the parent class. An association relationship may represent a uni-directional or bi-directional relationship between two objects and for the case of bi-directional relationship it is represented by a line. An aggregation relationship, on the other hand, is a specialised form of association in which a whole is related to its part. Aggregation is also known as a “part of” relationship. It is denoted in UML by a line, connecting the “part” and the “whole”, with a diamond next to the class showing a whole.

The three main relationship types are employed in *ACRU2000* to describe the interactions between various classes or objects. However, in order to help the reader appreciate the applicability of these relationships in conceptual physical based models, a simplified example is presented in Figure 4.3 that depicts the concept of objects and relationships as applied to the soil system and associated objects.

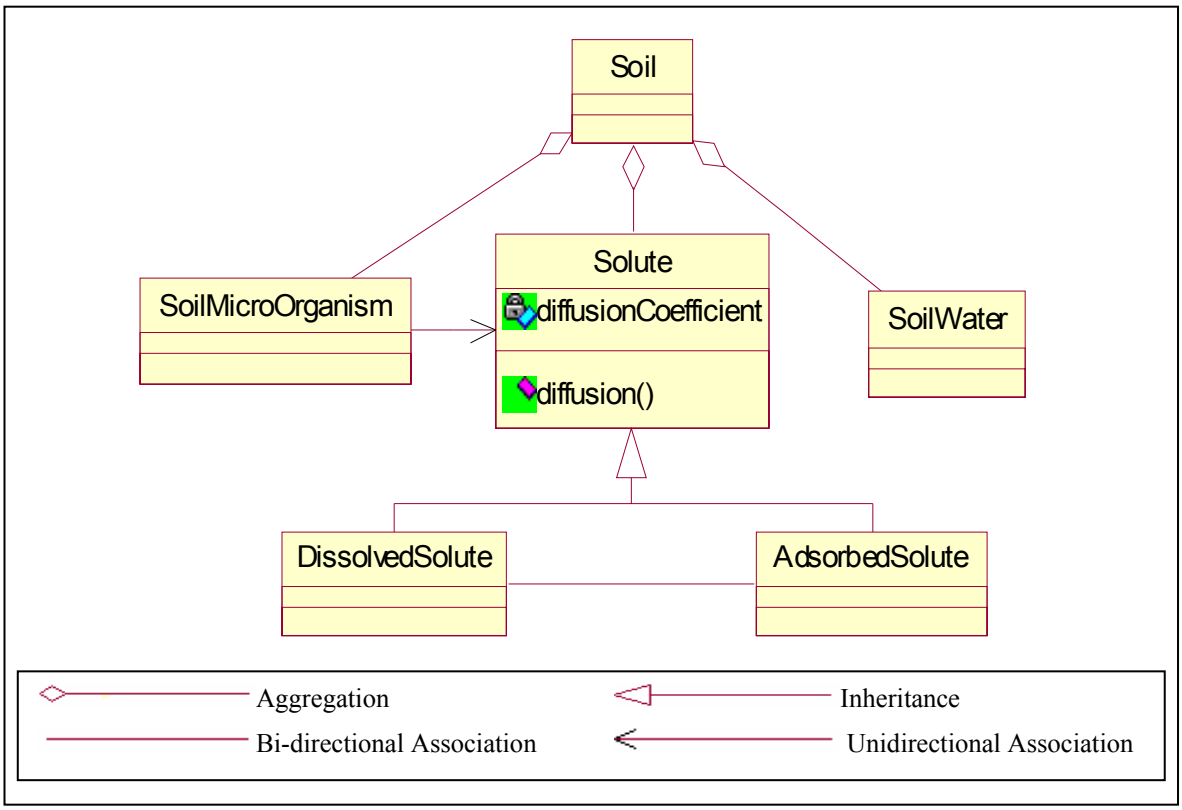

Figure 4.3 An example of objects and their relationships as applied in a simplified soil system

The soil object, among other objects, is comprised of soil microorganisms, solutes and soil water that are represented in Figure 4.3 by SoilMicroOrganism, Solute and SoilWater classes respectively. Thus, these three objects are "part of" the soil object and their relationship with the "whole" (soil object) is represented by an aggregation. A solute, in turn, can be either dissolved, in the form of soil solution or adsorbed on the soil particles. Therefore, depending on their mode of appearance and other associated characteristics, there are two "types of" solute. Hence, the DissolvedSolute and AdsorbedSolute classes are linked to the Solute classes by an inheritance relationship. Like all other objects a solute has attributes and

behaviours that characterise the object. As it is depicted in Figure 4.3, the Solute Class owns "diffusionCoefficient" and "diffusion" which respectively are its attribute and behaviour (operation). Thus, the two child classes (DissolvedSolute and AdsorbedSolute), being linked with their parent class (Solute) by an inheritance relationship, will be able to inherit the attribute and behavior of the Solute Class. From the diagram it can be seen there also exists an interaction between the SoilMicroOrganism and Solute as well as between DissolvedSolute and AdsorbedSolute classes. The interaction between these classes is an association. Solutes can change from the dissolved phase to an adsorbed phase, and vise versa, through the operation of diffusion. This movement of solutes, however, depends on the concentration gradient. Thus, in order for a solute to diffuse from one phase to another, these two objects need to exchange information regarding the solute load and concentration of both sides. Hence, the interaction between these two objects is described as a bi-directional association. Assuming a Solute Class does not need any information from the SoilMicroOrganism Class, while the SoilMicroOrganism needs information on the amount and state of a solute, then the relationship between these two objects can be represented by a unidirectional association.

4.3.2 Basic structure and objects of *ACRU2000*

Since *ACRU2000* is written with an object-oriented programming language, it is composed of Objects. There are seven main objects that constitute the model (Table 4.1). According to Kiker and Clark (2001), four of these objects: Model, Control, Interface and Exception operate out of sight to developers of new module. The Model Object starts model simulation through paving the way for other model objects. The Control Object, on the other hand, is responsible for managing the input and output systems of the Model. The Interface Object is used in *ACRU2000* for grouping of similar objects. The Exception Object handles errors that can emanate from conflicts between the new object under development and objects that belong to the programming language (Java). Such errors can result, for example, through a division by zero. It can also be used to handle errors that can arise due to the non compliance of the new object with the requirements already set by another *ACRU2000* object. For example, if a new data object is trying to add or deduct a value to or from a data object whose upper and lower value limits are already set, then the Exception object would handle and send an error message that the quantity requested by the object under development is out of the limits specified by another existing object. The remaining three objects, v*iz.* Components, Processes and Data, are the most important objects as far as modelling of hydrology is

concerned (Clark *et al.*, 2001). Most of the coding phase of this research project is also based on these three objects. Therefore, this section will attempt to describe these objects with the help of an example.

Table 4.1 Basic objects in *ACRU2000* (after Campbell *et al.*, 2001)

Object Name	**Example Classes in *ACRU2000***
Model	MAcru2000Extensible
Control	AAcru2000ModelInput, AAcru2000ModelOutput
Interface	IWaterFlow, INutrientCycle, ISaltFlow
Exception	EArithmeticException, EFileNotFoundException
Components	CLandSegment, CReach, CDam, CChannel
Processes	PSCSRunoff, PMUSLEErosion, PSaltInput
Data	DArea, DPrecipitation, DBaseflowSalinity

In order to maintain model consistency and for ease of differentiating which class belongs to which object, a convention has been adopted in *ACRU2000* that all class names should start with a capital letter (which is also the Java language convention) indicating the class type. Thus, all classes that belong to Component Object start with C, Process classes start with P and Data classes start with D (Clark *et al.*, 2001). The Control object, however, is an exception to this rule. Instead of starting with the letter C its object classes start with A. This was done to avoid confusion that could arise due to having the same starting letter as the Component Object. Further, Butler (2001) has used italic letters to show class names in his dissertation. Therefore, this dissertation will also adhere to all the above conventions so as to maintain model consistency both in the coding and documentation of *ACRU2000.*

According to Clark *et al.* (2001), the Component objects represent the physical component of the hydrological system being modelled and form the building blocks of *ACRU2000*. Objects such as *CVegetation*, *CClimate* and *CLandSegment* are examples of the Componet Objects (Figure 4.4). These objects, in turn, may contain other smaller objects that together constitute the whole. The *CVegetation* Object, for example, is composed of *CLeafCanopy*, *CStem* and *CRoots* objects. The term "land segment", represented as a *CLandSegment* Object, in *ACRU2000* replaces the term sub-catchment in *ACRU 300 Series.* This is done in order to

show that an area need not be self contained hydrologically and could be created from a digital elevation grid (Kiker and Clark, 2001).

Each Component Object contains data objects that can store and describe attributes inherent to the component. For example, as shown in Figure 4.4, the *CClimate* object contains *DPrecipitation* and *DRainfallSaltLoad* data objects. The *DPrecipitation* stores information pertaining to daily depth of precipitation and the *DRainfallSaltLoad* stores the quantity of salt associated with wet atmospheric deposition (rainfall salt load). According to Clark *et al.* (2001), data objects can also perform additional functions such as range checking and specification of data units.

The Process Object, as its name implies, is responsible for describing processes that can take place in a conceptual or real world hydrological system. Thus, process objects shown in Figure 4.4, such as *PSurfaceFlow*, *PSubsurfaceFlow* and *PGroundwaterFlow*, along with other associated process objects, describe the flow of water from the soil surface through soil horizons to groundwater store and runoff. Similarly, process objects other than those responsible for water flow do exist in *ACRU2000,* such as those responsible for the transport of sediment and nutrients from one component to another. For example, the *PSaltGeneration* Process (in the hydrosalinity module of *ACRU2000*) describes an increase in salinity level of soil solution and groundwater store because of the addition of salts as a result of the weathering processes acting upon the surrounding soil and rock materials.

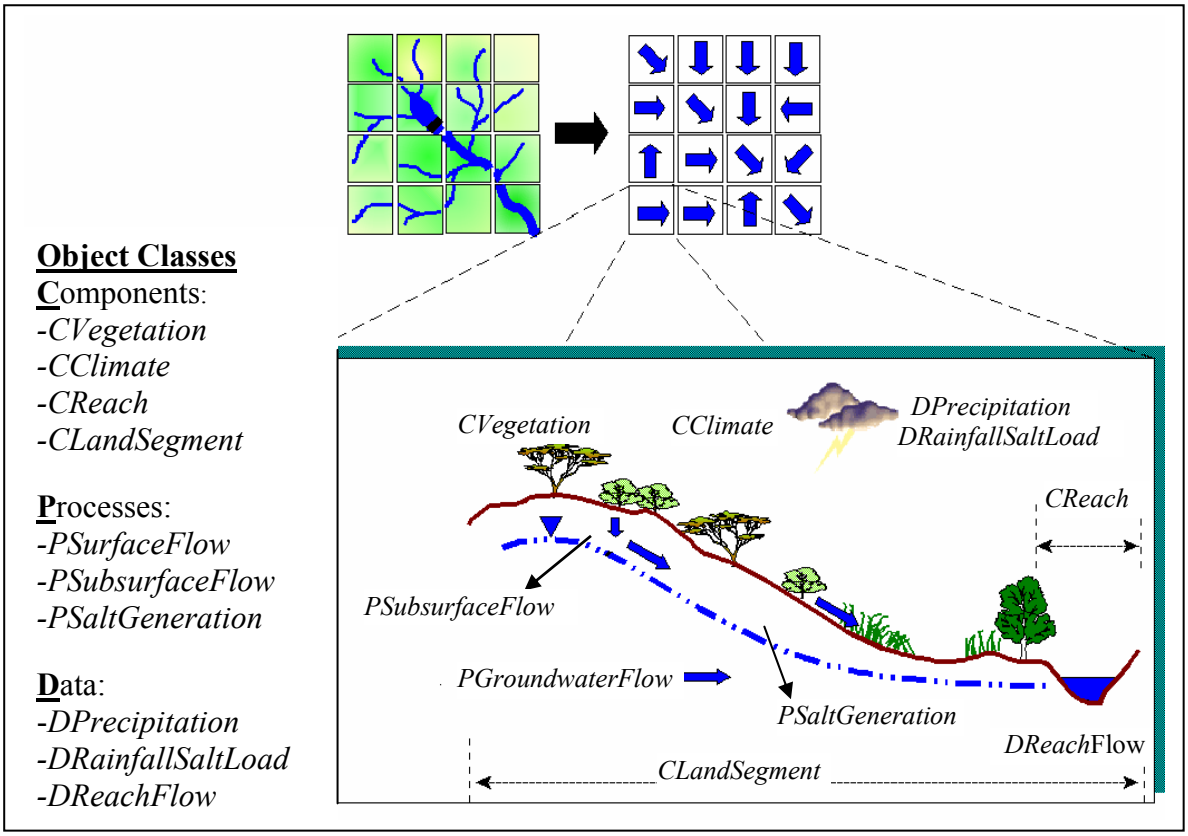

Figure 4.4 Examples of object classes as conceptualised in *ACRU2000* (after Clark *et al.*, 2001)

The hydrosalinity module of *ACRU* is developed within the *ACRU2000* environment. Therefore, it inherits the basic objects and structure of *ACRU2000* that have been described in this chapter. The following chapter deals with development of the hydrosalinity module of *ACRU* with special emphasis on description of the processes and associated data and component objects involved in the module.